Enterprise Integration

**Principles Statement**

Version: 1.1

Date: 28th February 2021

****Authors and Peer reviewers****

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

****Document Version History****

**Related Documents**

|  |  |  |
| --- | --- | --- |
| Document Name | Document type | Link |
| GT-ELT - AXA Cloud Usage Binding Principles - Public Version - v1.9.pdf | Stable | [Here](https://axa365.sharepoint.com/:b:/r/sites/Architecture/Shared%20Documents/Strategy/Cloud%20Binding%20Principles/GT-ELT%20-%20AXA%20Cloud%20Usage%20Binding%20Principles%20-%20Public%20Version%20-%20v1.9.pdf?csf=1&web=1&e=x4pEFW) |
| Sec Instructions - Network Segmentation v1.8.pdf | Stable | [Here](https://axa365.sharepoint.com/:b:/r/sites/onesecurity/Shared%20Documents/7.%20Security%20Instructions/01.%20Security%20Instructions%20-%20Oct.2020/Sec%20Instructions%20-%20Network%20Segmentation%20v1.8.pdf?csf=1&web=1&e=ddciv0) |
| Sec Instructions - Encryption v1.4.pdf | Stable | [Here](https://axa365.sharepoint.com/:b:/r/sites/onesecurity/Shared%20Documents/7.%20Security%20Instructions/01.%20Security%20Instructions%20-%20Oct.2020/Sec%20Instructions%20-%20Encryption%20v1.4.pdf?csf=1&web=1&e=uYxFsJ) |
| 2013 AXA Group Target Architecture v2 01.doc | Stable | [Here](https://axa365.sharepoint.com/:w:/r/sites/Architecture/Business%20Architecture/AXA%20Group%20Target%20Architecture/2013%20AXA%20Group%20Target%20Architecture%20v2%2001.doc?d=w5396667fc5e34bf38a09b02f55fcb28f&csf=1&web=1&e=Xmgkqf) |
|  |  |  |

****Table of Contents****

[1 Intended Audience 5](#_Toc96948954)

[2 Introduction 5](#_Toc96948955)

[3 Enterprise Integration Principles 5](#_Toc96948956)

[**PRINCIPLE 1** Promote real time integration wherever possible 5](#_Toc96948957)

[**PRINCIPLE 2** Make sure that integration is Observable 6](#_Toc96948958)

[**PRINCIPLE 3** Mediate API integration by default. 7](#_Toc96948959)

[**PRINCIPLE 4** Facilitate Asynchronous integrations 8](#_Toc96948960)

[**PRINCIPLE 5** Prefer an Application Centric approach 8](#_Toc96948961)

[**PRINCIPLE 6** Make interface independent from its implementation 9](#_Toc96948962)

[**PRINCIPLE 7** Catalogue integrations 10](#_Toc96948963)

[**PRINCIPLE 8** Federate with AXA approved IAM or other relevant IAM services 11](#_Toc96948964)

[**PRINCIPLE 9** Adopt and support the AXA Zero Trust approach 11](#_Toc96948965)

[**PRINCIPLE 10** Promote Self-service and apply least privileged delegated management to integration flows owners. 12](#_Toc96948966)

# Intended Audience

The primary audience for Enterprise Integration Reference Architecture is the IT community, chiefly, the Enterprise, Platform and Solution Architecture communities, but it is also relevant beyond the architecture community: cloud brokers, developers, Infrastructure managers, Program and project managers will all benefit from gaining an understanding of the Enterprise Integration principles.

# Introduction

This document contains the list of stated principles to be applied to AXA’s Enterprise Integration domain for all new integration implementations.

In the scope of this document the use of the terms “**Integration**” and “**Integration Flow**” are interchangeable, and we defined them as:

the set of tools, infrastructure, practices, and software artefacts (including, but not limited to, the interfaces and implementations of REST API’s, Queues, web Services, etc.) used to establish the communication and exchange of data and services between two or more applications

# Enterprise Integration Principles

Below is the set of principles that should govern the implementation of all future integration requirements.

In addition to the principles documented here, all integration flows need to comply with the **GAR Fundamental Principles**, **Cloud Security Binding Principles** (as specified by group Security) and **AXA Governance Principles**.

1. Promote real time integration wherever possible

DESCRIPTION

Service providers and consumers should, whenever possible, use real, or near-real, time integration to expose and access capabilities from other services.

To cope with the use cases where real-time is not feasible or advisable, the actual integration mode should be based on the list of published integration design patterns correspondent to the identified and catalogued use cases.

RATIONALE

This principle will promote real-time integrations, as opposed to traditional batch processing, where sometimes it’s necessary to wait to create a daily batch to transfer hundreds of records. It supports the increasing need for straight-through-processing, improved customer experience especially in context of digitalization.

IMPLICATIONS

**When possible, avoid File Based Integration:** File based integrations should be avoided whenever possible, when there are other alternatives available that are better aligned with the real-time principle. In certain situations, such as bulk data loads, it may make sense to use file batches, but even then, alternatives should be considered if available (for instance, bulk messaging, data streaming or using a de-batch pattern).

**Use Batch Integration only when it makes sense:** this principle does not preclude the use of batch integration, but in principle batches should only be implemented when their benefits outweigh the alternatives.

1. Make sure that integration is Observable

DESCRIPTION

It should be possible to observe via monitoring, with end-to-end traceability capabilities from integration source to destination over all integrations, and as close to real time as possible.

RATIONALE

The observation of integration flows supports several requirements amongst which:

* **security**: detecting abnormal traffic patterns (volumes, frequency, distribution, location...) contribute to proactively react to threats,
* **quality of service**: tracking consumption, response time, failed attempts, etc. and automating alerts help improve the service performance and meet SLAs,
* **usage**: understanding how the integration is consumed can fuel a feedback loop and drive its future roadmap, targeting consumers with proper communication, also helps to identify abuse when compared to the SLA of the respective consumer
* **costing**: be it for internal financial purposes or for consumer monetization, knowing who is using what can bring a lot of value. Depending on what is at stake, it can go from near real-time to delayed monitoring. Monitoring data can also be subject to various persistency policies.

While the capture of the monitoring raw data can be done in different ways and places depending on the integration pattern and deployment, all these inputs must be easily accessible and retrievable to be consolidated in an enterprise monitoring/reporting tool.

IMPLICATIONS

**Real-Time Monitoring:** Continuous monitoring and governance of events. This implies having observation events that react to monitoring events, such as alerts when error limits are reached. And end-to-end detailed traceability, from the event source to destination.

**Universal Unique Transaction Identifiers:** The system must provide a feature to identify a single transaction or a group of transactions from numerous transaction logs.

**Consider Mediate Integration:** All API’s and events should be mediated; point-to-point integration should be avoided. This is the consequence of the implementation of many other principles listed here. Indeed, there are many benefits expected from this mediation layer which are even greater the earlier in the processing chain they can be delivered, for instance:

* **security**: central policy enforcement
* **performance**: they can be optimized through the mediation layer (routing, caching, failover, etc), so that the sooner it happens the better it is,
* **monitoring**: it should be as close as possible from what the consumer is experiencing,
* **decoupling**: it should be easier to change endpoints, etc

Moreover, this mediation layer must be under the control of the enterprise, not delegated to any third party.

1. Mediate API integration by default.

DESCRIPTION

Mediation is the way to manage the API experience with a component between consumer and provider of an API. This layer currently is called “API experience management” as quite often a one-size-fits-all approach to API design and exposure doesn’t work, and different types of users, developers, and devices have different expectations and requirements when it comes to API consumption, this layer becomes appropriate and useful

RATIONALE

API is diverse in terms of needs from consumers and providers when signing the API contract, so a mediation layer as an experience management layer gives the opportunity to:

* **Change in integration requirements**: APIs are built out to serve a single use case, product or business line, but implementing this layer provides lots of constituencies each with different use cases and integration requirements.
* **Response to consumer needs**: Existing API layer has been in product for several years and needs to be enhanced with new functionality DevOps teams ask for.
* **Data governance**: Synchronize data across a variety of services — even if they are in different domains. With many departments within an organization making their own purchasing decisions for the products they use; central control and data governance can be lost.
* **Scalability over time**: Integrate to a particular service now, but in the future need to swap this for a new product or connect to multiple products.
* **Hide complexity**: For objects or resources that exist in multiple underlying applications, databases or other sources and want to provide consistent access to these as API resources to shield the consumer of the resource from complexity.
* **3rd party app integration expectations**: bringing a digital business application to market, and customers of this application will expect integration to the SaaS apps they use within their organization.

IMPLICATIONS

**~~Prefer stateless integrations and APIs:~~** ~~Service statelessness as main principles to be applied for scalability, designing scalable services by separating them from their state data whenever possible. This results in a reduction of the resources consumed by a service as the actual state data management is delegated to an external component. By reducing resource consumption, the service can handle more requests in a reliable manner.~~

1. Facilitate Asynchronous integrations

DESCRIPTION

This principle means that the integration associated action should be processed independently from the integration flow that triggers it. When required, the outcome can be retrieved through a subsequent integration flow.

RATIONALE

Building asynchronous integration has the benefit of improving and can, very often, be processed near real-time.

For instance, issuing a policy may only require the consumer to know that his/her request is confirmed/approved, or being processed, which can be delivered synchronously then leaving more time to proceed with the chain of subsequent actions (creating the policy potentially with subsequent checks, maybe the customer, generating the documentation).

These various steps can be executed with limited to no delay in near real-time, but as they are independent the integration benefits from more flexibility to deal with issues, and to adjust to processing constraints on the consumer’s side.

IMPLICATIONS

**Asynchronous integrations should be preferred**: unless there’s an absolute need for blocking a consumer pending the outcome of a request, and when it’s desirable to have the absolute minimum overhead and time of response (e.g., a web page loading). Note that an integration flow can be asynchronous but still perform in near-real-time, and thus this principle should not be in contradiction with the real-time principle.

**Helps decoupling applications:** asynchronous integrations help decoupling applications and reduces interdependencies. Both ends of an integration flow (i.e., both applications/systems interfaces) should be designed and implemented to maximise their reusability.

1. Prefer an Application Centric approach

DESCRIPTION

Application owners should be responsible for choreography and data transformation (enrichment, mapping, etc.), within their own application scope.

RATIONALE

The use of integration facilitators “near” application components reduces latency and bandwidth requirements. Additionally, this simplifies operational models and lifecycle management processes, promoting higher responsibility and control on a single team, as owner of the application.

IMPLICATIONS

**Follow the SDLC of the applications:** Integration development should follow the development life cycle of the applications.

**Prefer application adapters before Technology adapters before custom integrations**: application integrations should choose application adapters before technology adapters before direct (adapter-less) integrations. Application adapters (SALESFORCE, SAP) simplify integration by providing easy access to the business objects and business functions provided by the application. Technology adapters (JSON, FTPS, ETC) are widely reusable, but provide no application-specific help to integrate to an application. Direct connections to applications (SDKs) provide no opportunity for reuse and are likely to break with any change to the application.

1. Make interface independent from its implementation

DESCRIPTION

There must be a clear separation of concerns between the interface of an integration and its actual implementation (or implementations). In addition to this separation the interface of an integration should also be independent of the infrastructure used to support its implementation.

RATIONALE

This principle makes it easier to decouple providers and consumers of the integration, not only in terms of the usage of the integration (i.e., “run time”) but also in terms of the development lifecycle, making it possible for the developments to run in parallel or independently between the producing and consuming side.

The interface is the only component the consumer should care about, it should abstract how the service is implemented. The interface and the service should support disconnected lifecycle where changes on one side can be transparent on the other side. The same interface can be used to access several services implementations or instances, each of which potentially delivered through different ways or technologies, deployed on various infrastructures. Portability is more and more a topic when dealing with software engineering, integration must support it.

IMPLICATIONS

**Infrastructure agnostic Integration Interfaces:** The design and management of the integration must be independent from the infrastructure. Typically, the consumer of an integration does not need to know about the infrastructure on which the provider is running it. A change in the infrastructure should not impact the interface thus being transparent to the consumer.

**Contract first integrations:** Any integration must be borne by a contract between the consumer and the provider. This contract must define what is made available by the provider (purpose and specification), under which conditions/SLAs (availability, performance, commercial terms, support...) and how it should be used by the consumer. Such a contract must be made available/known to the consumer as part of the interface and as such independent from the implementation. Of course, any change to the contract must be made aware in due time to the consumer.

**Catalogued Integration assets (e.g., API’s):** Well Defined, Standardized & Discoverable assets (API’s, Events, etc.) create more stable integrations, limits the impacts of changes, and facilitates reuse.

**Loosely Coupled Integration provider and consumer:** When dealing with distributed systems integration should design interactions to be loosely coupled and whenever possible, asynchronous to manage problems such as availability and latency. This will also promote reusability (future consumers will be able to tap into existing produced services or events), and event driven architectures.

**Hybrid deployment:** It should be possible to deploy integration components in a timely manner either on premises or in a variety of CSP’s, with components to collect/receive events close to their source/destination.

**Avoid single landing zone designs:** Avoid designs for single deployment mode only.

**Design for reversibility:** Facilitate the reversibility of applications and integrations deployments.

1. Catalogue integrations

DESCRIPTION

Each integration must be referenced in a catalogue. This reference should act as an inventory of all available integrations with a minimum set of characteristics aiming to describe both the purpose and usage. It is not to be mixed with integration portals aiming to engage with targeted consumers.

Cataloguing also starts in design mode and it’s part of the architecture best practices.

RATIONALE

This referential is the cornerstone of the integration’s management and governance all along its lifecycle. One can only manage what is known. Integrations are exposed to increasing security threats and shadow integrations (i.e., not catalogued) are a predominant risk.

Failing to have a comprehensive catalogue may lead to deficiencies like:

* security risks related to unknown integrations,
* unmanaged technical debt by lack of tracking,
* duplication of similar integrations if what is existing is not properly discoverable,
* poor quality of service due to unproper documentation of dependencies,

Catalogue capabilities may be offered natively by the implemented integration platforms which may lead to a distributed – or even worse siloed – documentation in case several solutions or instances are deployed. While some of the requirements may still be satisfied in such a case, a consolidated view supporting central management must be made available. Especially, it must ensure that all integrations at stake to support a specific use case can be easily identified and managed no matter their patterns.

IMPLICATIONS

**Consistency in integration management:** integrations should be pattern based, these patterns should be catalogue for easier discoverability and reusability, and all produced integration assets should be registered to promote reusability and facilitate self-service.

**Integration exposure should be consistent**: even though integration flows can diverge on certain aspects (e.g., internal vs external) they should be exposed in a consistent manner, for instance through a unified catalogue with a well-defined RBAC model.

1. Federate with AXA approved IAM or other relevant IAM services

DESCRIPTION

Integrations must use approved AXA identity and access management (e.g., AXA Global IAM, AXA Entities’ local IAM, or approved third party IAM), for authentication and authorization, enabling Role Based Access Control.

RATIONALE

Users and clients must be authenticated with an approved AXA IAM (e.g., Global or Entity) before granting integration system resource access rights.

IMPLICATIONS

**Verify all requests:** All incoming messages must be verified to ensure they come from authenticated and authorized system resources.

1. Adopt and support the AXA Zero Trust approach

DESCRIPTION

Integration flows must adopt the **AXA Zero Trust Approach** (i.e., don’t trust anything, verify everything) supporting its implementation as part of end-to-end flows.

RATIONALE

The integration layer is a fundamental aspect of the Zero Trust Approach as it sits in between the applications and can thus enforce security policies and ensure that the communication between applications is performed through secured, audited channels.

IMPLICATIONS

**Don’t rely on network security:** Integrations should not rely only on network security (IP whitelisting) and perimeter network controls. Network security (IP whitelisting) as Auth schema and secure Access policy can be used as a second factor for Authentication process, but never the only one.

**Federated Access Control:** For access control, applications should delegate the authentication via federation protocols.

**User Rights Provisioning:** For user identity, account, and coarse-grained rights (roles) lifecycle management and golden source rely on IAM systems. IAM systems can provision various endpoints to sync IAM data.

**Integrations should be secured by design and security managed:** As per the ability to run in multiple environments and provide integration across corporate boundaries such as partner integration, security risks need to be managed. Authentication, authorization, and vulnerability protection policies need to be addressed in integration design. Every integration should implement authentication and authorization regardless of their network location

1. Promote Self-service and apply least privileged delegated management to integration flows owners.

DESCRIPTION

The integration infrastructure should facilitate self-service to both consumers (register as consumer of X API, Event, etc.) and providers (register a new API, Event, etc).

RATIONALE

The usage of an existing integration should be self-service enabled. There is obviously a lot of value to reuse existing integrations rather than duplicating them: it reduces implementation, maintenance, and operation costs, as well as speed of delivery and limit complexity. Any constraint going against self-servicing is a barrier to reuse and must be avoided. Also, with self-servicing comes the need to make integration capabilities accessible (i.e., available and user friendly) to whoever can be Involved not assuming that they need upfront to build/acquire specialized skills.

IMPLICATIONS

**Requires RBAC model:** self-service needs to be managed or restricted based on an RBAC model to ensure that different users can use the available capabilities with different levels of access and proper rights.

**Segregation between domains:** self-service does not mean that everyone will have access to everything, so there will be a domain segregation that together with the RBAC policy will restrict access to different domains to different groups of people.

**Requires integration portal:** consumer self-servicing support will require an integration portal, available and accessible by the consumers.

**Integrate with CI/CD pipeline:** providers will need to integrate the catalogue with their CI/CD pipeline.

**People and Process:** Leveraging modern practice and capabilities like DevOps, event-driven architecture, container-based infrastructure, distributing the responsibility for creating and maintaining integrations beyond the central team.

**Role of Central Team:** even though this principle aims at promoting self-service, some key responsibilities will have to remain the responsibility of a central team, like:

* Governance
* Maintaining shared documentation such as this document, patterns, etc.
* Facilitating the communication between entities
* Foster the adoption of shared standards and best practices