



(REVIEW ARTICLE)



API-FIRST ENTERPRISE ARCHITECTURE FOR SAP DIGITAL TRANSFORMATION

PURNACHANDER RAO NADIPELLI *

IT Technical Manager | ERP Transformation | SAP S/4HANA, BTP - Corning Inc.

* Corresponding Author

ORCID Details

PURNACHANDER RAO NADIPELLI: <https://orcid.org/0009-0000-7616-3015>

World Journal of Advanced Research and Reviews, 2022, 13(02), 683–700

Publication history: Received on 21 January 2022; revised on 24 February 2022; accepted on 27 February 2022

Article DOI: <https://doi.org/10.30574/wjarr.2022.13.2.0203>

Copyright © 2022 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

Successful enterprise digital transformation is becoming more and more about integration between core business systems, cloud platforms, applications, and outside organizations. This study aims to define an API-first enterprise architecture in the context of integration between SAP S/4HANA, SAP Business Technology Platform (BTP) and external enterprise systems. A design-oriented, literature-based approach was followed for the synthesis of the research, technical standards, security guidance, and SAP documentation within reference period parameters. This proposed architecture also includes contract-first design, API management, security, lifecycle governance, monitoring, version control – with API governance assumed to be the fundamental boundary for integration. Based on the analysis, it has been found that API-first architecture offers the advantages of decreasing system coupling, enhancing interface reusability, establishing interoperability, and exerting a stronger control over enterprise contract. But in order to be implemented successfully, organizations need additional elements like versioning, API retirement, monitoring, authentication and authorization, and ownership. The study adds to the available information an integrated reference architecture which correlates the API design principles with SAP specific integration requirements and enterprise governance. The findings indicate that API-first can be a good fit for a SAP digital transformation that treats APIs as a governed, repeatable, and reusable enterprise capability as opposed to a technical endpoint.

Keywords: API-first architecture; SAP S/4HANA; SAP BTP; Enterprise architecture; API management; Digital transformation; API governance; Enterprise integration.

1 INTRODUCTION

As more and more enterprises go through digital transformation, they need to integrate their essential business systems with cloud applications, digital channels, partners, and new technologies. This suggests the need for integration architectures which enable business capabilities to be exposed in a controlled and reusable way. Application programming interfaces (APIs) offer this sort of architectural mechanism where they define particular interfaces for communications among pieces of software without relying on the provider's internal implementation which consequently reduces a reliance of consumers on the provider's internal technicalities (Dudjak & Martinović, 2020).

The API-first design idea goes a step further by designing the API contract as a first-class architectural design ingredient, and then implementing the software. The OpenAPI Specification (OAS) defines a standard, machine-readable way to describe an HTTP API, its operations, parameters, data structure and its responses as well as security requirements (OpenAPI Initiative, 2020). This allows API contracts to be present for API development, documentation, testing and communication between API providers and consumers.

The approach is especially relevant in the context of an SAP environment. SAP marked S/4HANA to become the digital core of the Intelligent Enterprise and APIs/Business events as enabler mechanisms of consumption automation and integration. SAP also announced the investment in an API-first approach to software development that is designed to offer better access to the cloud functionalities of S/4HANA and enable third-party integration (SAP, 2020). So APIs can offer a controlled API boundary between S/4HANA business capabilities and applications running throughout the enterprise ecosystem.

But enterprise API strategy is more than just providing endpoints. APIs need to be designed securely and need to be governed, monitored, versioned and also retired. According to NIST, there are several factors to bear in mind when working with API based distributed architectures, such as authentication, access management, secure communications, monitoring, resilience, load balancing, and throttling (Chandramouli, 2019). Technical interfaces need to be paired with governance and lifecycle-management practices, therefore, to make API-first architecture work.

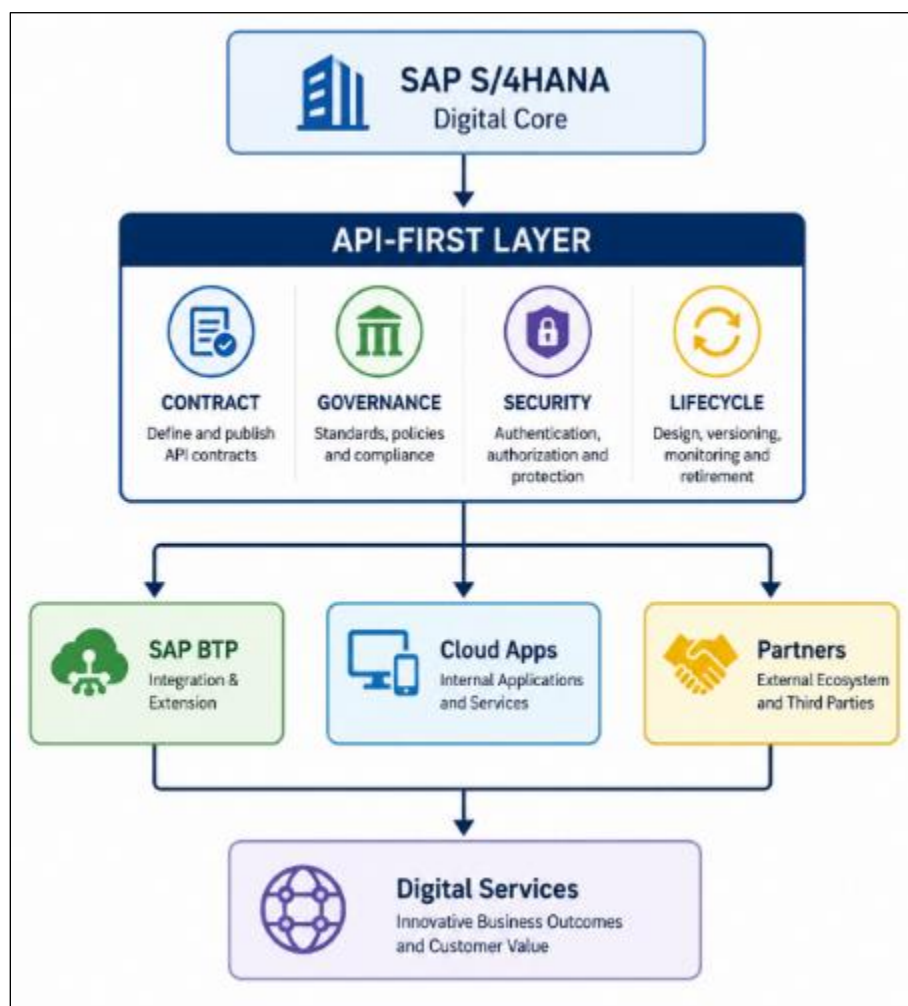


Figure 1: Conceptual foundation of API-first SAP digital transformation

Source: Author's synthesis based on SAP (2020), Dudjak and Martinović (2020), OpenAPI Initiative (2020), and Chandramouli (2019).

1.1 Problem Statement

As enterprises add apps and services, traditional point-to-point integration can continue to grow in numbers. SAP integration that is based on direct connections between S/4HANA and any number of consumers can complicate integration changes, and it can tempt an organization to create multiple integrated interfaces and integration logic.

By implementing an API-first approach and creating standardized interfaces around business capabilities, direct dependencies can be reduced. But APIs alone aren't an effective API ecosystem. If the governance is not being done properly, there might be inconsistencies in the designs, different forms of ownership, proliferation of API's, incompatible versions, poor security, poor lifecycle management, etc. Research about the API economy also shows that API adoption and value are not one and the same, and that organizational and governance issues impact successful API programs (Simon, 2021).

Thus, the problem of this study is how an API-first enterprise architecture could be designed and implemented to facilitate secure, reusable, interoperable and maintainable integration of the SAP S/4HANA, SAP BTP and other enterprise systems.

1.2 Research Objectives

The purpose of this research is to establish and analyse an API-first enterprise architecture (EA) for the digital transformation of SAP. In particular, it aims to:

- Investigate the API-first principles and how they will affect applications and environments in SAP;
- creating the reference architecture between S/4HANA, SAP BTP and external systems;
- Review API governance, API lifecycle-management needs;
- Examine enterprise APIs security measures; and
- Understand the advantages and limitations of API First Integration vs traditional integration.

1.3 Research Questions

- **RQ1:** How will API first take greater supporting role for digital transformation in an SAP in center enterprise?
- **RQ2:** What are possible ways in which APIs can assist integration between SAP S/4HANA and SAP BTP as well as external systems?
- **RQ3:** Which governance, lifecycle and security features are needed for a sustainable enterprise API ecosystem?
- **RQ4:** What advantages and disadvantages are associated with adopting API-first architecture over or instead of the traditional point-to-point integration?

1.4 Contribution of the Study

The study provides an integrated reference architecture of API-first design, with integration to SAP, governance, security and lifecycle management. Instead of seeing only as a technical endpoint API, it sees API as an enterprise asset that needs to be governed and undergoes standardised contracts, controlled access, reuse, and managed evolution. The final architecture forms the backbone for anyone who wants to leverage APIs to engage SAP digital-core functionality with any larger ecosystem of cloud applications, partners and enterprise services.

2 LITERATURE REVIEW

2.1 Enterprise Integration and API-First Architecture

Enterprise integration has progressed, moving away from the tightly coupled application connections, and towards architectures with modularity, interoperability, and independently managed services. APIs enable components in a distributed system to request capabilities from each other, without requiring any priori knowledge of its implementation. This separation is pertinent to microservice-oriented architectures (MOAs) where the communication boundaries between different independently developed microservices need to be stable (Dudjak & Martinović, 2020).

API-first architecture is an extension of this concept; its design centers around the API contract. The API was designed with the contract in mind first, functionality to follow. The marriage of API first and modular service design and testing is illustrated by Dudjak and Martinović (2020) in the design of a Backend-as-a-Service (BaaS) platform as a collection of microservices.

The OpenAPI Specification is a key technical commodity that enables contract-oriented development. It represents a programming language agnostic, standardized way of describing HTTP APIs, where API operations, parameters, and request and response data formats along with security constraints are described in a machine-readable format (OpenAPI Initiative, 2020). This allows you to document, build, test and educate consumers with a single API specification.

2.2 APIs and the API Economy

APIs are also evolved from a technical integration mechanism. API economy is a term that refers to digitally exposing capabilities and linking organizations, services, applications and external ecosystems with APIs. In her post on the API economy throughout her career, Simon (2021) further explores the shift towards the API economy, outlines the strategy behind APIs, and challenges the notion that API adoption has an automatic economic benefit.

An important difference for enterprise architecture. There are no guarantees that a big API portfolio will equate to good digital transformation for an organization. Business value is important to consider when assessing whether APIs expose meaningful capability, that the quality and usability are sufficient, appropriate consumers are reached, and the APIs are governed during their lifecycle (Simon, 2021).

This means that on the API level of the SAP it's all about business capabilities, not all about APIs. APIs should be picked and shaped based on integration needs and the goals of the business, and not as a way to "expose everything available on the backend."

2.3 SAP S/4HANA API Ecosystem

SAP's API strategy provides a platform-specific foundation for applying API-first principles. SAP identified S/4HANA as the digital core of the Intelligent Enterprise and described APIs and business events as mechanisms for consumption, automation, and integration. SAP also stated that its API-first development strategy was intended to provide greater access to S/4HANA Cloud capabilities and support third-party integration (SAP, 2020).

The SAP API Business Hub further supports API discovery and consumption by making SAP APIs available through a centralized ecosystem (SAP, 2020). This creates several characteristics relevant to API-first architecture: discoverability, reuse, standardized access, and separation between consuming applications and backend capabilities.

The SAP context therefore complements the general API-first literature. While OpenAPI establishes a standardized method for describing APIs and academic research establishes the architectural rationale for contract-first development, SAP provides a concrete enterprise environment in which these principles can be applied to ERP business capabilities.

2.4 API Management and Security

Apart from the API contract, API first architecture needs management specifics. APIs in distributed deployments need to be authenticated, to have access controls, secure communication, monitoring, be resilient, and be capable of traffic-management. NIST deems these important considerations to be for microservice based applications, and API gateways are architectural mechanisms that can be used to implement several cross-cutting capabilities (Chandramouli, 2019).

APIs are potentially useful for accessing business operations or enterprise data, so security is a prime concern. A framework governing access to a protected resource is being defined by the use of an access token, an owner delegate from the resource owner to a service provider. OAuth 2.0 defines an authorization framework that involves the use of access tokens, which serve as an owner delegate to a service provider to access particular protected resources (Hardt, 2012). This can be used together with API gateway and identity-management mechanisms to form the foundation for a controlled API access (Chandramouli, 2019).

With API management also comes governance necessities. With the proliferation of APIs in organizations' portfolios, there must be ways to document, own, version, monitor, and retire APIs. In the absence of such controls, API ecosystems can be spread and hard to manage. This is to reinforce the difference between API Exposure and API Governance.

Table 1: Literature synthesis

Literature area	Key finding	Relevance to this study
API-first development	API contracts support modular distributed architectures (Dudjak & Martinović, 2020)	Establishes API-first foundation
OpenAPI	Standardized machine-readable API descriptions (OpenAPI Initiative, 2020)	Supports contract management
API economy	API adoption requires organizational and strategic considerations (Simon, 2021)	Supports governance perspective

SAP API strategy	S/4HANA APIs support consumption and third-party integration (SAP, 2020)	Establishes SAP-specific context
API security	Distributed APIs require authentication, access control and monitoring (Chandramouli, 2019)	Supports security architecture
OAuth	Access tokens provide a standardized authorization mechanism (Hardt, 2012)	Supports API authorization

Source: Author's synthesis of the reviewed literature.

2.5 Research Gap

There is strong literature that covers individual aspects of API-first enterprise architecture. API-first research touches on contract-oriented development, OpenAPI answers how to describe an API in a standardized manner, NIST answers the question of the security of distributed APIs, and both OAuth answers the question of API authorization and SAP documents APIs for S/4HANA integration (Dudjak & Martinović, 2020; OpenAPI Initiative, 2020; Chandramouli, 2019; Hardt, 2012; SAP, 2020).

But such opinions are often discussed one at a time. The specific gap this study will focus on is: A single SAP-oriented architectural model that includes API-first design, SAP S/4HANA integration, governance, security, lifecycle management, and enterprise interoperability.

The study will fill this gap by consolidating the evidence found into an architecture for reference and assessing it based on these criteria: reuse, interoperability, security, scalability and lifecycle governance.

3 RESEARCH METHODOLOGY

3.1 Research Design

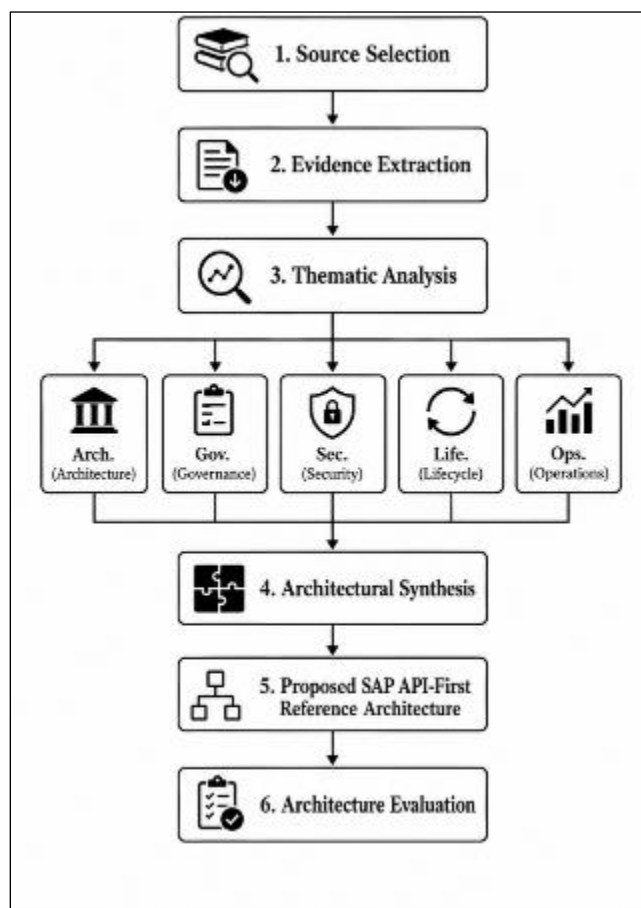


Figure 2: Research methodology

Source: Author's methodological framework.

I have taken the design-oriented Literature Based Research Approach to achieve the API First enterprise Architecture for SAP Digital Transformation. Instead of trying out every software system on the market, the study brings together existing research, technical standards, security advice and SAP documentation, and builds on and tests a reference architecture.

The methodology involves 4 stages, selection of sources, analysis of evidence, synthesis of architecture and evaluation.

3.2 Source Selection

References cited in the study were published in 2021 or prior as a part of the research requirement. Sources are chosen based on the relevance to API-first architecture, SAP integration, API management, security, governance, enterprise architecture.

There are four categories of sources taken into account:

- Academic writings on API-first development and API economy;
- Technical standards such as OpenAPI, OAuth;
- Technical guidance independent of users, including NIST security guidance; and
- SAP documentation – proof related to S/4HANA APIs and SAP's approach towards APIs.

Multiple source categories provide the flexibility of less reliance on vendor literature and still have the evidence required for S/4HANA-centric architecture.

Table 2: Source-selection criteria

Criterion	Requirement
Publication date	2021 or earlier
Relevance	Direct connection to research objectives
Authority	Academic, standards, government, or authoritative SAP source
Technical applicability	Supports an architectural component or finding
Traceability	Verifiable bibliographic information
Evidence contribution	Supports at least one research question

3.3 Thematic Analysis

Five dimensions are used for analysing selected sources:

- Architecture: API First, Contracts, Abstraction and Integration;
- Governance – Standards, Ownership, Discovery and Versioning;
- Security: authentication, authorisation, access control and monitoring;
- Lifecycle: design, publication, evolution, deprecation and retirement; and
- Operations: Scalability, Resilience, Analytics and Monitoring.

Findings from different sources will be combined but not merely summarised literature with the thematic approach.

3.4 Architectural Synthesis and Evaluation

Five analytical dimensions are translated into a multi-layered SAP API-first architecture with consumers, API management, SAP S/4HANA, SAP BTP, and external enterprise systems.

Proposed architecture is checked against five criteria – reusability, interoperability, security, scalability, and governance. Reusability asks whether the API can be used by several customers, Interoperability asks about the problems with standardizing the interaction with heterogeneous systems, security asks about access and protection mechanisms, scalability asks about the ability to support the increasing requests for integration, and governance asks about control over the API lifecycle.

The resulting architecture is thus an evidence-informed reference model and not a statement that all SAP implementations have to be based on the same topology. To find architectural principles and relationships that can be used in various SAP enterprise.

4 PROPOSED API-FIRST ARCHITECTURE

4.1 Architectural Model

Proposed architecture place APIs between the applications / partners / services, consuming APIs and business capabilities within the SAP context. This is a model which introduces the API first' paradigm in which interfaces should offer clear boundaries for service providers and consumers (Dudjak & Martinović, 2020). OpenAPI also fosters this by providing an opportunity to formally describe API contracts in a language and technology-independent manner (OpenAPI Initiative, 2020).

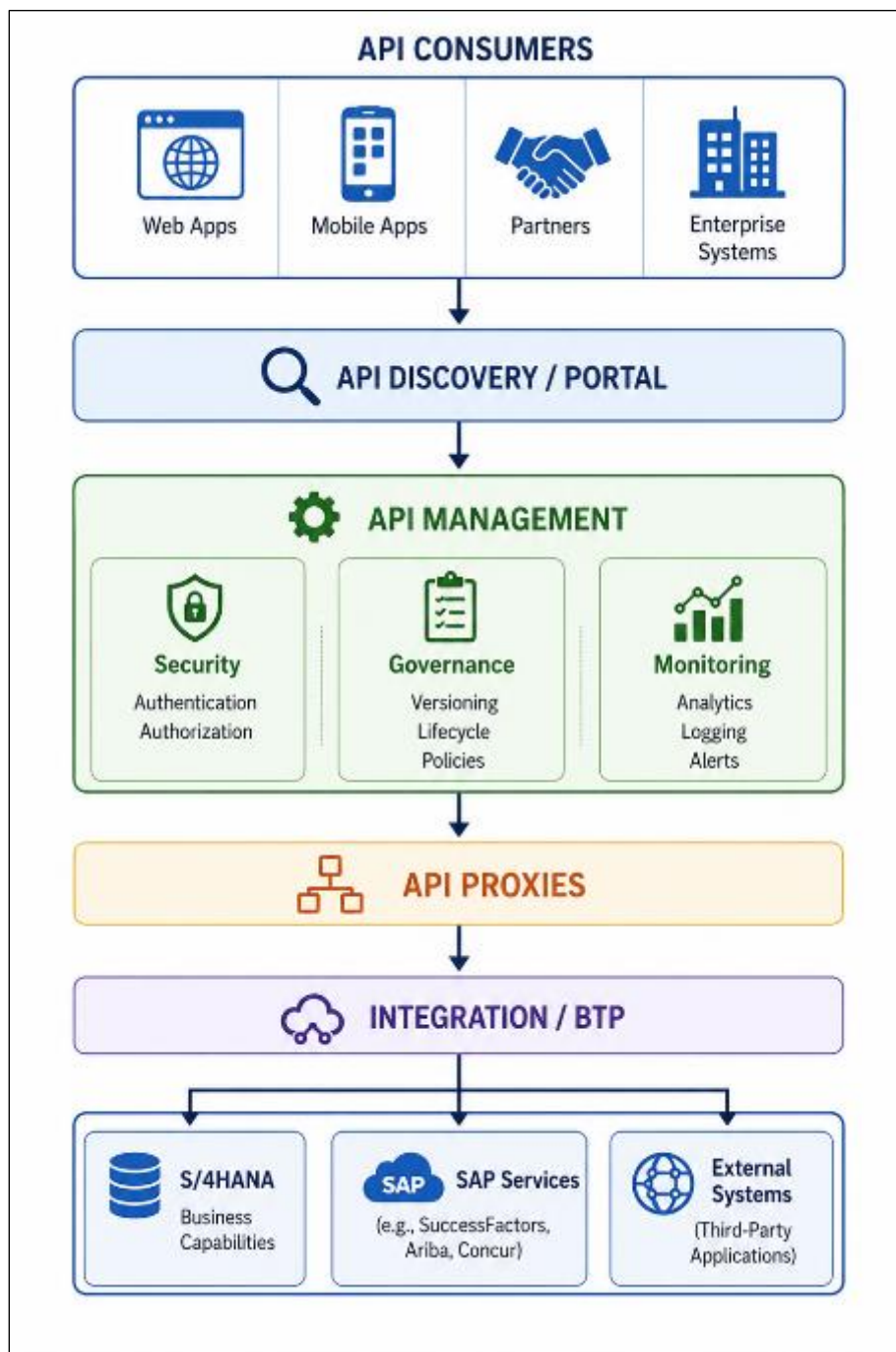


Figure 3: Proposed SAP API-first enterprise architecture

Source: Author's synthesis based on SAP (2020), Dudjak and Martinović (2020), OpenAPI Initiative (2020), and Chandramouli (2019).

This model is built on the foundation of SAP's API strategy and the platform-specific implementations. SAP identified APIs and business events as portals for consumption and integration around S/4HANA and presented an API-first development strategy to allow consumption of S/4HANA Cloud capabilities and third-party integration (SAP, 2020).

The designed architecture has 5 layers as follows:

- Consumer Layer– Web, Mobile, Partner and Enterprise Applications,
- API management layer — API discovery, security, policies, monitoring and lifecycle controls;
- Integration / extension layer: SAP BTP / services;
- Business-system layer: SAP S/4HANA and core enterprise capabilities; and
- External-system layer — third party applications/services in the cloud.

The architecture is a **proposed synthesis** rather than an assertion that SAP prescribes this exact deployment topology.

4.2 Consumer Layer

The consumer layer is made up of systems and applications that demand enterprise capabilities. This can encompass web applications, mobile applications, partner systems, in-house applications and other enterprise services.

Consumers should not need to know about the backend implementation structures, but rather deal with published API contracts. The importance of defining interfaces in API-first development is establishing boundaries between independently developed components (Dudjak & Martinović, 2020).

This is a mechanism that enables a number of consumers to share the same business capability. A customer-management feature, for instance, provided via an API might help multiple applications while avoiding the need for various applications to have their own affinity to it.

This is done by introducing API-management and integration layers between the consumer layer and the backend.

4.3 API Management Layer

It is possible to conclude that the API-management layer is located in the middle of the proposed architecture which serves as the central control point. It is not just for routing requests but also has the role of security, governance, monitoring, discovery and lifecycle management.

NIST defines API gateways as architectural mechanisms in distributed service architectures that can provide authentication, access management, secure communication, security monitoring, service discovery, resilience, load balancing and throttling operations (Chandramouli, 2019).

Thus, the proposed layer has five big functions:

Security: Before access to protected services is granted, authentication and authorization controls are in place. OAuth 2.0 lets a standardised authorization scheme for access tokens (Hardt, 2012).

- Governance: API standards, ownership, versioning and lifecycle policies are uniformly applied.
- Discovery: API descriptions, documentation means that authorised consumers can detect the services provided.
- Monitoring: Analytics and logging provide the ability to observe the behavior and API usage.
- Lifecycle management: APIs could be managed during publication, as well as evolution, deprecation and retirement.

These functions are explicitly called at the architectural level so that the implementation of common controls can be avoided at the individual application consuming level.

4.4 SAP S/4HANA Business-System Layer

We can see that S/4HANA is at the heart of the business system position since SAP recognizes and names it as the Digital Core of the Intelligent Enterprise, and offers APIs with which the capabilities of S/4HANA can be used and integrated (SAP, 2020).

The internal ERP implementation is not directly accessible by consumers based on the proposed architecture. In the service of this instead, business capabilities are made accessible via governed interfaces.

This establishes an abstraction boundary, in two ways. First, consumers rely on published contracts not on the implementation details that are behind it. Second, security and governance controls can be applied prior to requests getting to the core system.

The format of the APIs being published by SAP via the SAP API Business Hub is an indicator of the consumption patterns of SAP's wider API-based pattern surrounding S/4HANA (SAP, 2020).

4.5 SAP BTP and Integration/Extension Layer

SAP BTP is located in the middle between the core business capabilities and the vast application landscape. This gives a separation between the ERP digital core and other services or applications.

The relevance of this separation lies in API-first philosophy on decreasing direct links between service consumers and providers (Dudjak & Martinović, 2020). APIs can deliver a managed interaction point between applications and business capabilities, enabling further services to be built on-top of the core point of interaction.

The layer can thus be used in integration and extension scenarios where applications utilise SAP capabilities but don't necessarily "contain" all of the custom functionality directly inside the ERP core.

4.6 External Enterprise Systems

In most modern companies, there are systems not in the SAP environment. Third-party integration was specifically called out by SAP in their stated API-first vision as an important driver for offering their S/4HANA Cloud capabilities on APIs (SAP, 2020).

With the proposed architecture, the external systems are thus first-class members in the enterprise API ecosystem. SAP APIs can be consumed by external application types and APIs provided by other external systems can also be consumed by SAP oriented application types.

To enable this heterogeneous environment, the OpenAPI's language-independent description model allows consumers to gloss over and comprehend an API regardless of the programming language that it is implemented in (OpenAPI Initiative, 2020).

4.7 Request and Integration Flow

A representative interaction starts when an accepted member of society distinguishes an API and sends a request. The Api-management layer takes care of validation of authentication and authorization requirements, application of relevant policies, and ramping the request through an appropriate Api-proxy to the backend service.

According to NIST, authentication and access management are critical attributes of API gateway architectures (Chandramouli, 2019) and OAuth 2.0 is a standardized authorization mechanism that relies on access tokens (Hardt, 2012).

Table 3: Architectural layers and responsibilities

Layer	Primary responsibility	Key architectural concern
Consumer	Requests business capabilities	API consumption
API discovery	Provides API information	Discoverability
API management	Controls API access and operation	Security and governance
API proxy	Mediates consumer-backend interaction	Abstraction
SAP BTP/integration	Supports services and integration	Extension/interoperability
S/4HANA	Provides core business capabilities	Business-system integrity
External systems	Provide or consume external capabilities	Heterogeneous integration

Source: Author's synthesis based on SAP (2020), OpenAPI Initiative (2020), and Chandramouli (2019).

4.8 Architectural Benefits

Several potential benefits are possible with the proposed architecture.

API contracts enable controlled dependency between the API consumer and API provider (Reduced coupling) (Dudjak & Martinović, 2020).

Reusability – One business capability can serve several consumers instead of having new integration logic for each (SAP, 2020).

Centralized security: API gateways have a centralized place for authentication, access management, and monitoring (Chandramouli, 2019).

Interoperability: Standardized API descriptions can facilitate the interactions across different technological heterogeneous environments (OpenAPI Initiative, 2020).

Explicit contracts bring forward visibility to changes, and enable structured version and lifecycle management.

4.9 Architectural Challenges

The architecture also brings with it some compromises.

The API-management layer can become an important infrastructural dependency, as centralising security and policy enforcement can simplify governance. NIST thus promotes the use of proper architectural and security control of gateway components (Chandramouli, 2019).

API proliferation is also an issue. An enterprise can build a multitude of APIs without delivering any meaningful business capabilities that can be reused. Simon (2021) believes that API adoption shouldn't be equated to business value necessarily.

Thus, governance is pivotal. If a portfolio is to be coherent, then APIs need to be owned, documented, put to version, add security controls and have lifecycle processes.

This suggested architecture should thus be seen as a means to manage integration complexity and not remove it.

4.10 Summary

It is about proposing the APIs as the major integration interface between SAP S/4HANA and the broader enterprise. API management offers capabilities for centralized security, governance, discovery, monitoring and life cycle management, and SAP BTP and integration services offer an environment for extensions and connections to core business capabilities.

The architecture is designed to be API First, includes SAP-specific integration needs, and lays the groundwork for discussing API design, governance, security, and lifecycle management in the next sections.

5 API DESIGN AND CONTRACT MANAGEMENT

5.1 Contract-First Design

API-first architecture considers the API contract a fundamental design document and designates the interactions between the API's consumers and its providers before actual implementation. This makes it modular and that services are clearly defined in a distributed architecture (Dudjak & Martinović, 2020).

The OpenAPI Specification is a standardized description of the operations, parameters, data structure, responses and security requirements of an HTTP API in a machine-readable format (OpenAPI Initiative, 2020). In the case of a SAP system, this facilitates a consumers' dependency on S/4HANA implementation details and also offers consumer re-usable interfaces for various systems.

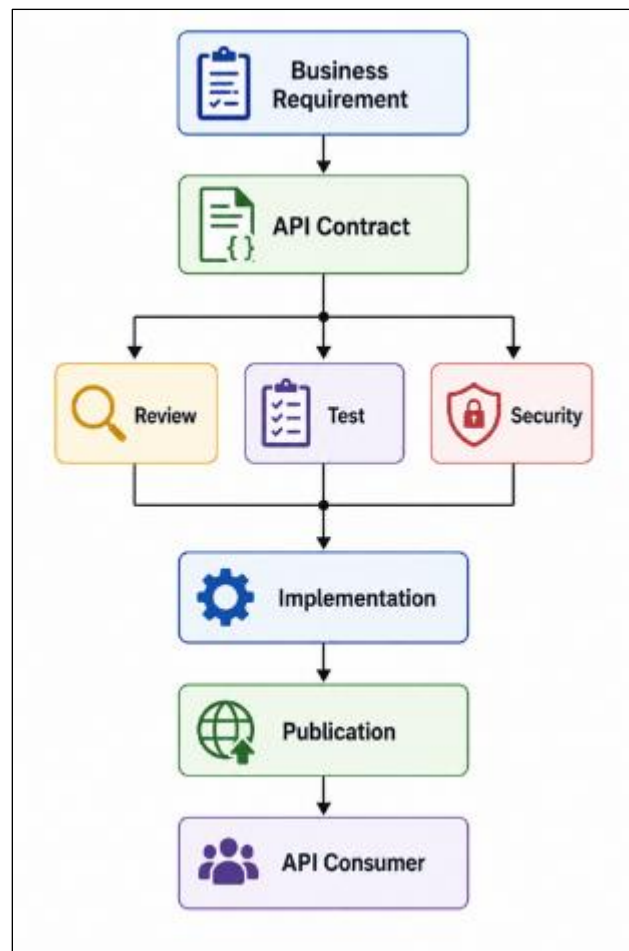


Figure 4: Contract-first API development

Source: Author's synthesis based on Dudjak and Martinović (2020) and OpenAPI Initiative (2020).

5.2 Standardization, Abstraction, and Reuse

OpenAPI provides a language-independent API description which helps facilitate interoperability (OpenAPI Initiative, 2020). SAP's API-first strategy also aims to make more of the capabilities of S/4HANA accessible and facilitate third-party integration (SAP, 2020). API-first is more of a mindset and governance than a protocol requirement, as S/4HANA can expose capabilities based on OData, and other technologies such as SOAP can be supported.

APIs should be thinking at the meaning of the business capability, not simply mirroring backend structures. This abstraction leads to lesser coupling, while enabling reusability across many consumers (Dudjak & Martinović, 2020).

Table 4: Key API design principles

Principle	Purpose	SAP application
Contract-first	Predictable interfaces	S/4HANA API contracts
Abstraction	Reduce implementation dependency	API mediation
Reuse	Support multiple consumers	Shared business APIs
Standardization	Improve consistency	OpenAPI standards
Security	Protect API access	Authentication/authorization

Source: Author's synthesis based on Dudjak and Martinović (2020), OpenAPI Initiative (2020), and SAP (2020).

5.3 Versioning, Discovery, and Validation

Since APIs are shared between applications, these contracts need to be carefully managed when it comes to changes. This paper suggests two types of changes: compatible and breaking changes and controlled versioning and migration

strategies for breaking changes. Too many active versions should be avoided as there is management and maintenance overhead.

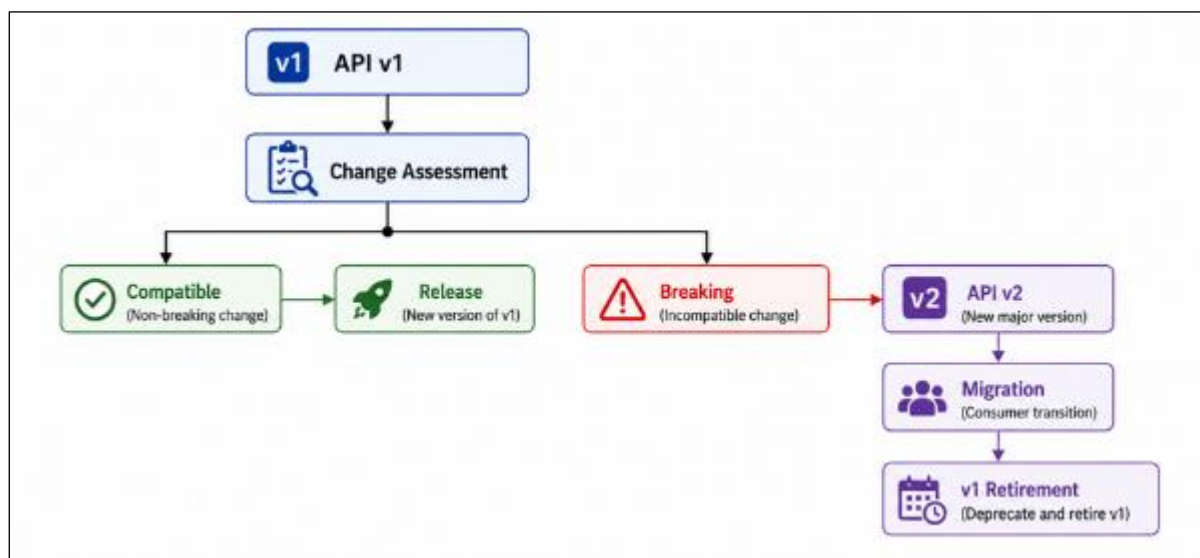


Figure 5: API version management

Source: Author's synthesis based on Dudjak and Martinović (2020).

Discoverability and documentation are also key to API reuse. In addition, APIs are discovered with the help of SAP's API Business Hub, and OpenAPI supplies machine-readable API descriptions for documentation and development tooling support (SAP, 2020; OpenAPI Initiative, 2020). Contract specifications can also help to measure and test implementation and validation of the contract (Dudjak & Martinović, 2020).

5.4 Summary

Clear contracts, abstraction and standardization, reuse, documentation, version control, and validation are all key design elements for an effective SAP API. They create well-defined interfaces to S/4HANA capabilities and consuming applications and form the basis for the governance and lifecycle controls discussed in Section 6.

6 API GOVERNANCE AND LIFECYCLE MANAGEMENT

6.1 API Governance

The goal of API governance is to define the organizational and technical controls that are needed to help the enterprise API portfolio achieve consistency. APIs face challenges related to API lifecycle management, security, versioning and standards when they are used as shared enterprise assets, requiring the need for API governance. Governance is part of the API maturity since API adoption alone cannot ensure business value.

Governance is concentrated on ownership, standards, security, Life Cycle Management and monitoring for the proposed SAP architecture.

Table 5: Proposed API governance framework

Area	Responsibility	Example control
Ownership	Establish accountability	Named API owner
Standards	Ensure consistent design	API guidelines
Security	Control access	Authentication/authorization
Lifecycle	Manage evolution	Version/deprecation policy
Monitoring	Track usage and operation	Logs/analytics

Source: Author's synthesis based on Simon (2021), Chandramouli (2019), OpenAPI Initiative (2020), and SAP (2020).

6.2 API Lifecycle Management

Governance should cover API design all the way to their retirement. OpenAPI simplifies publishing, consuming, monitoring and managing APIs by its standardized approach to API definition, whereas SAP's API-management approach are the mechanisms to support this in SAP.

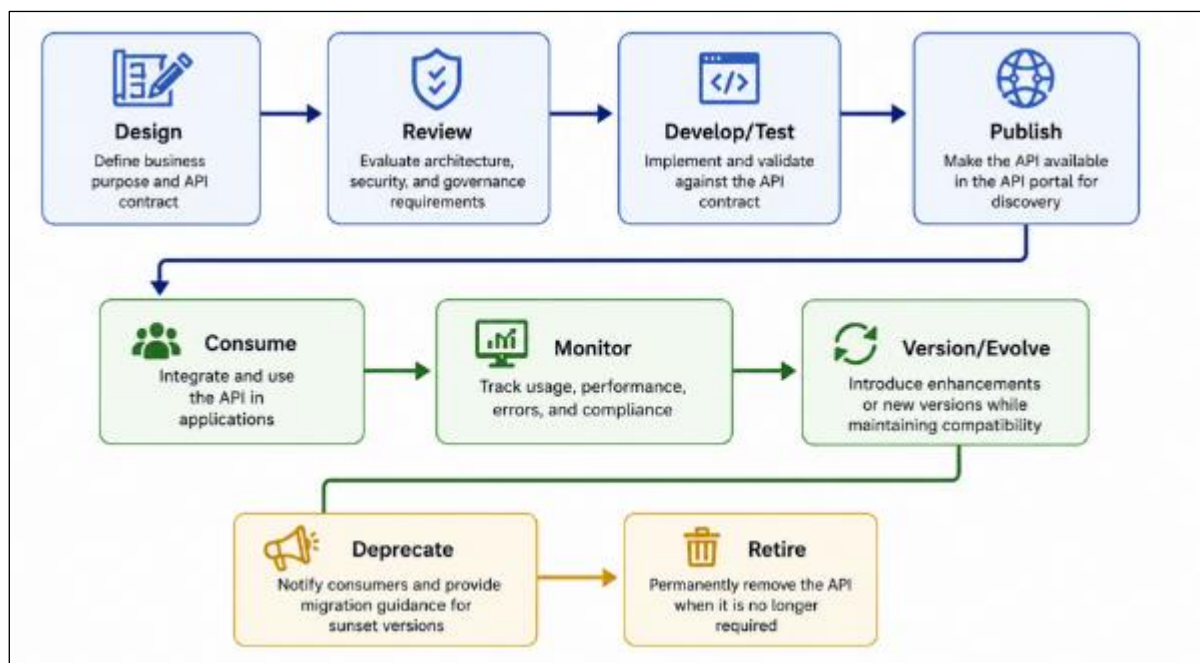


Figure 6: Proposed API lifecycle

Source: Author's synthesis based on OpenAPI Initiative (2020), SAP (2020), and Simon (2021).

Each stage has a specific purpose: design establishes the business capability and contract; review evaluates architecture, security, and governance; development and testing validate implementation; publication and consumption make the API available; and monitoring and evolution support continued operation. When an API is no longer required, formal deprecation should precede retirement.

6.3 Versioning, Monitoring, and Governance Challenges

Version management is important because if you make a change in the API it can impact several API consumers. Organizations should therefore pinpoint breaking changes, tell consumers about it and offer suitable migration routes as well. Deprecation offers a gradual approach to retirement.

Monitoring will not only assist operations but can also provide information on API usage, failures, traffic, and performance for governance. In distributed API environments, NIST defines several relevant capabilities, such as monitoring, resilience, load balancing, and throttling (Chandramouli, 2019), and SAP offers API analytics and monitoring capabilities (SAP, 2020).

Governance needs to be proportional, too. Too many approvals can slow down development, and inadequate governance can lead to duplicate and conflicting APIs. This study therefore suggests a risk-based governance model that represents the intensity of governance by the criticality of the API, scope of API consumers, security needs, and impact of integration.

6.4 Summary

API governance is more than just turning individual APIs into a collection of managed interfaces; it's the process of creating a business-wide API portfolio. The controls for sustainable API adoption are ownership, standards, security, lifecycle management and monitoring. The proposed lifecycle gives APIs governance, from design, to usage, to retirement.

7 API SECURITY ARCHITECTURE

7.1 Security Requirements

Security is also key to enterprise API architecture, as APIs can be used to share business processes and data with internal and external consumers. According to NIST, some pertinent requirements for API-based distributed architectures include (Chandramouli, 2019): authentication, access management, secure communication, monitoring, resilience, and traffic management.

The proposed SAP architecture is indeed defense-in-depth, with security measures spread over three distinct tiers: API management, identity infrastructure, communication channels, and backend services.

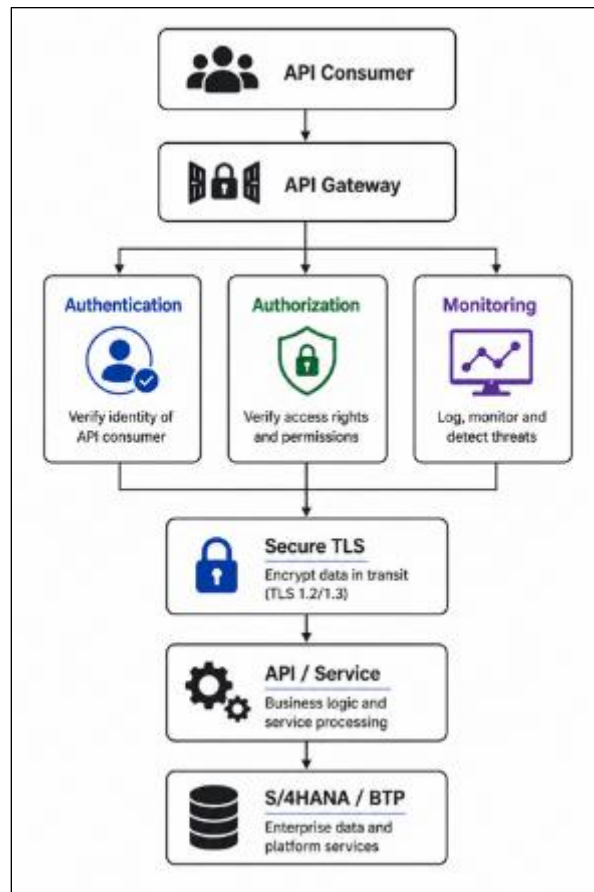


Figure 7: Proposed API security architecture

Source: Author's synthesis based on Chandramouli (2019), Hardt (2012), and SAP (2020).

7.2 Authentication, Authorization, and Gateway Controls

Authentication is the process that verifies who is requesting, then authorization makes decisions about what resources and/or operations can be accessed. OAuth 2.0 is a standard authorization protocol that uses access tokens to gain access to protected resources (Hardt, 2012). According to NIST, API gateways are also defined as "a way to add authentication, access management, monitoring, secure communication, and traffic control into a distributed system" (Chandramouli, 2019).

Therefore, the proposed architecture is that the API gateway sits in front of the services running on SAP and common policies are applied upstream of S/4HANA. Gateway security, however, must be used in conjunction with, but not as a replacement for, backend security, so that if a gateway is compromised, enterprise resources are not necessarily exposed.

7.3 Secure Communication and Monitoring

Secure communication shields API traffic between consumers, API management, integration services and backend systems. TLS can be used on these communication paths, and consistent with NIST's focus on secure communication in distributed API designs (Chandramouli, 2019).

Monitoring gives you visibility of authentication failure, denied requests, unusual traffic, errors in the APIs and service availability. SAP's API-management features also enable monitoring and analytics to assist in operational visibility (SAP, 2020).

Table 6: Key API security controls

Security area	Objective	Example control
Authentication	Establish identity	Identity provider
Authorization	Control access	OAuth 2.0/access tokens
Communication	Protect data in transit	TLS
Gateway	Apply common policies	API gateway
Monitoring	Detect abnormal activity	Logs/analytics
Traffic control	Limit excessive requests	Throttling
Backend security	Protect core services	Service-level authorization

Source: Author's synthesis based on Hardt (2012), Chandramouli (2019), and SAP (2020).

7.4 Security Challenges

Centralized API security helps to achieve uniformity, but adds extra dependencies. API gateway is an important infrastructure element that can impact the availability of many APIs and therefore resilience is of importance (Chandramouli, 2019). Too much permissions can also have a negative impact on authorizations, and monitoring data needs to be protected as well. Last, API, consumer, or capability changes should trigger the re-evaluation of security needs.

7.5 Summary

The proposed security model includes authentication, authorization, secure communication, gateway policies, monitoring and protection for the back end. OAuth 2.0 serves as the foundation for authorization, and NIST guidance recommends layered security controls and the use of API gateways (Hardt, 2012; Chandramouli, 2019). In the SAP environment, this allows for controlled access to S/4HANA and BTP and ensures security on various architectural layers.

8 INTEGRATION, SCALABILITY, AND RESILIENCE

8.1 Enterprise Integration

The proposed API-first architecture offers an integration barrier between SAP S/4HANA, SAP BTP, external applications, and enterprise consumers. APIs enable these systems to share business capabilities via well-defined interfaces, not just direct point-to-point connections. This helps to promote the modularity and loose coupling of API-based distributed architectures (Dudjak & Martinović, 2020).

Third-party integration with S/4HANA Cloud is clearly expressed in SAP's API-first approach, and APIs will be an important part of the ways to integrate SAP features with external applications (SAP, 2020). To ensure interoperability, OpenAPI also offers a standard way to describe HTTP APIs, regardless of the language in which they are implemented (OpenAPI Initiative, 2020).

8.2 Scalability

The architecture must be scalable, meaning that as the number of consumers and the number of service interactions increase there should not be too much dependency. NIST also recognizes that independent scalability is a key feature of microservice-based architectures and lays out some load balancing and throttling mechanisms that can be used to handle the distributed API environments (Chandramouli, 2019).

The proposed architecture allows for API management to act as a safe gateway for traffic to be monitored and controlled, and for backend services to be kept independent from the consumer's implementation. This enables further consumers to utilize the existing APIs without having to establish direct links to S/4HANA.

8.3 Resilience

Resilience is about the architecture's ability to function even when some components fail or the demand for services varies. In the context of distributed API architectures, NIST considers resilience and availability to be important (Chandramouli, 2019).

The proposed model enables separation between consumers, API management, integration services and backend systems, which allows for resilience. But, this division does not necessarily imply resilience. APIs, integration services and backend systems need to be monitored and protected from failures as well.

This study suggests that scalability and resilience should be measured on the entire API path instead of the API endpoint.

Table 7: Integration and resilience considerations

Area	Architectural mechanism	Expected outcome
Integration	Standardized APIs	Interoperability
Scalability	Load management and decoupling	Capacity for additional consumers
Traffic control	Throttling	Controlled resource usage
Monitoring	API analytics and logging	Operational visibility
Resilience	Layered architecture	Reduced dependency concentration

Source: Author's synthesis based on Chandramouli (2019), Dudjak and Martinović (2020), SAP (2020), and OpenAPI Initiative (2020).

9 DISCUSSION

The results prove that API-first architecture offers a structured way to connect SAP S/4HANA to cloud, enterprise applications, partners and digital channels. The proposed architecture is a consolidation of contract based interfaces, API management, security controls, governance and lifecycle management in one architecture. This is moving away from just an endpoint for integration to a governed enterprise resource.

The first research question was to address how API-first architecture can facilitate SAP digital transformation. The analysis shows that APIs can reveal reusable business capabilities and minimise direct dependencies of consuming applications on SAP backend implementations. This fits in with the API-first principle and with the microservices literature that focuses on explicit service boundaries and modularity (Dudjak & Martinović, 2020).

The second research question focused on integration with S/4HANA, SAP BTP and outside systems. In this context, SAP's API-first approach demonstrates how APIs can facilitate access to SAP S/4HANA's capabilities and integration with third parties (SAP, 2020). OpenAPI also offers a consistent way to describe HTTP interfaces using a variety of different technologies (OpenAPI Initiative, 2020).

The third research question was about governance and security. The results suggest that API publication is not enough. In distributed API environments (distributed systems) authentication, authorization, secure communication, monitoring, and traffic management are essential elements (Chandramouli, 2019; Hardt, 2012). Ownership, versioning, documentation, and retirement are other aspects of governance that need to be covered.

Last but not least, the fourth research question explored the advantages and difficulties of API-first integration. The advantages include: reduced coupling, reuse, interoperability, centralized controls and improved lifecycle visibility. Such benefits, however, add governance requirements and additional infrastructure dependencies. If these factors are not managed intentionally, they can jeopardize the benefits anticipated from the API. Without intentional management, API proliferation, gateway dependency and contract-management complexity can cause such problems.

In sum, the results lend support to implementing API-first architecture both as a technical integration strategy and as an enterprise governance strategy. The value of the architecture goes far beyond the number of APIs deployed; it comes down to the quality, security, reuse, and lifecycle management of APIs (Simon, 2021).

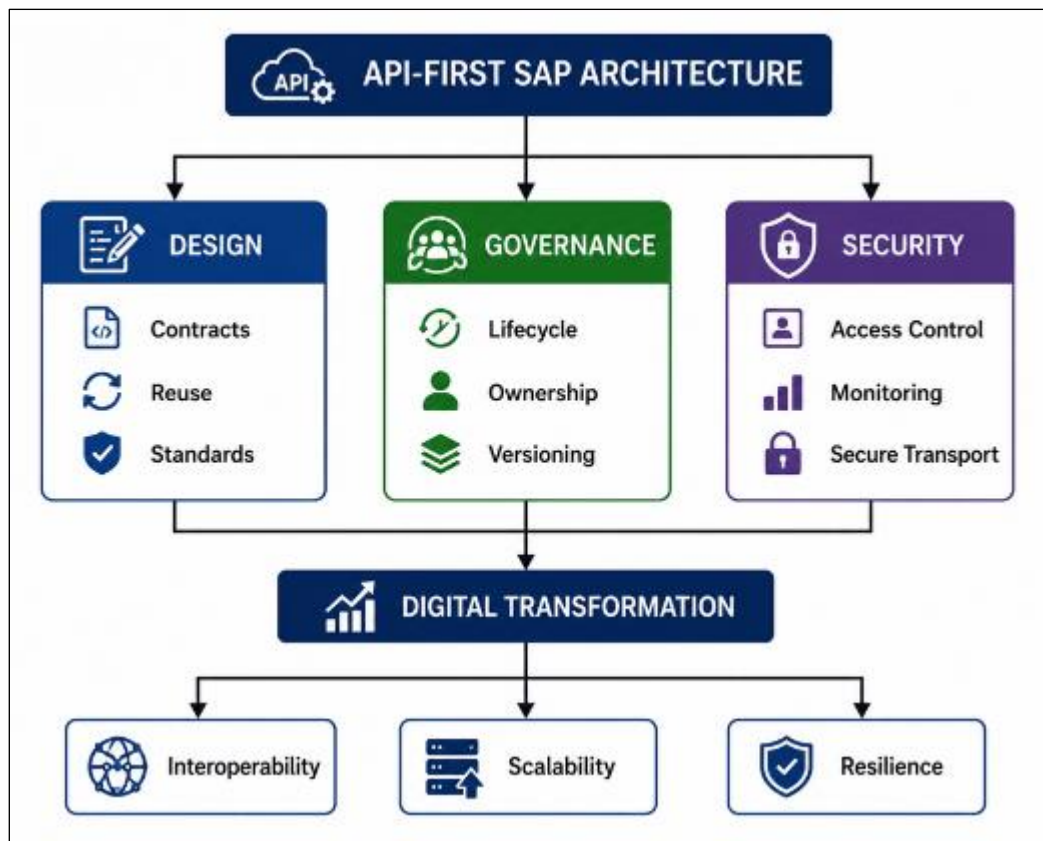


Figure 8: Summary of research findings

Source: Author's synthesis based on the findings of this study.

10 CONCLUSION

API-first enterprise architecture offers a structured way to connect SAP S/4HANA to applications, cloud services, partners and other enterprise systems. This study proved that the benefit of an API-first approach is not limited to the technical side of the connectivity: API can create interface for reusing business capabilities, decrease direct coupling, and offer a controlled integration boundary (Dudjak & Martinović, 2020; SAP, 2020).

The proposed architecture includes API contracts, API management, SAP BTP, S/4HANA, external systems, security controls, and lifecycle governance. OpenAPI offers a standardized means of defining an API contract and OAuth 2.0 and API gateway capabilities offer a mechanism for controlling access to protected services (OpenAPI Initiative, 2020; Hardt, 2012; Chandramouli, 2019).

Another important finding of this study is that API-first adoption brings governance responsibilities. From ownership and documentation to versioning and monitoring, security, and retirement, the management of APIs must be done systematically to avoid uncontrolled integration dependency and API proliferation. Therefore, technical infrastructure and organizational controls must work together for successful SAP API transformation.

The model suggested by the proposed framework is not an implementation, but rather it acts as a reference model. The architecture can be validated in the future using hands-on SAP deployments, and by quantifying its performance, scalability, integration effort and operational costs.

Finally, when APIs are perceived as enterprise capabilities that are reusable, secure, governed, continuously managed, API-first architecture can be an important basis for SAP digital transformation.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Disclosure of conflict of interest

The authors declare no conflicts of interest regarding this manuscript.

REFERENCES

- [1] Chandramouli, R. (2019). Security strategies for microservices-based application systems. National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-204>
- [2] Dudjak, M., & Martinović, G. (2020). An API-first methodology for designing a microservice-based Backend as a Service platform. *Information technology and control*, 49(2), 206-223.
- [3] Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. University of California, Irvine.
- [4] Hardt, D. (2012). The OAuth 2.0 authorization framework (RFC 6749). Internet Engineering Task Force. <https://doi.org/10.17487/RFC6749>
- [5] International Organization for Standardization. (2014). ISO/IEC 27001:2013 information technology—Security techniques—Information security management systems—Requirements. ISO.
- [6] OpenAPI Initiative. (2020). OpenAPI specification version 3.0.3. Linux Foundation. <https://spec.openapis.org/oas/v3.0.3>
- [7] Richardson, C. (2018). *Microservices patterns: With examples in Java*. Manning Publications.
- [8] SAP. (2020). SAP S/4HANA APIs and API-first development. SAP SE.
- [9] SAP. (2020). SAP API Business Hub. SAP SE.
- [10] SAP. (2020). SAP Cloud Platform integration and API management. SAP SE.
- [11] SAP. (2020). SAP Business Technology Platform: Architecture and integration. SAP SE.
- [12] Simon, Jean. (2021). APIs, the glue under the hood. Looking for the “API economy”. *Digital Policy, Regulation and Governance*. 23. 489-508. 10.1108/DPRG-10-2020-0147.
- [13] Souppaya, M., Morello, J., & Scarfone, K. (2019). Application container security guide (NIST Special Publication 800-190). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-190>
- [14] World Wide Web Consortium. (2018). Web Authentication: An API for accessing public key credentials level 1. W3C.
- [15] Zimmermann, A., Schmidt, R., Sandkuhl, K., Jørgensen, H. D., & others. (2019). Microservices and APIs: A systematic review of the state of the art. *International Journal of Computer Science and Information Technologies*.