



PUBLIC

2024-02-09

ABAP Cloud: Background Concepts and Overview

Content

1	Why ABAP Cloud?	3
1.1	About This Guide.	4
1.2	How ABAP Cloud Relates To...	6
1.3	Evolution Towards ABAP Cloud.	9
1.4	Relevant Personas in ABAP Cloud.	10
2	ABAP Cloud Development Model.	13
2.1	Built-In Qualities.	15
2.2	Key Concepts.	18
	Broad Use Case Coverage.	18
	Cloud-Optimized ABAP Language.	23
	Public Released APIs.	24
	Extensibility.	26
	Transactional Consistency.	28
2.3	Reuse Services and Libraries.	29
2.4	Tools Overview.	30
3	End-to-End Development with ABAP Cloud.	32
3.1	Design.	35
	Conceptual Background: Navigating the Foundations.	36
	Transactional Use Cases - Initial Considerations.	43
	Analytical Use Cases - Initial Considerations	45
	Integration Use Cases - Initial Considerations.	46
3.2	Develop.	49
	Develop SAP Fiori Apps.	51
	Develop Integration Services.	63
	Develop Extensions.	73
	Develop with Asynchronous Programming Patterns.	79
	Develop Localization and Internationalization.	81
	Document Apps and Services.	82
3.3	Test.	83
	Develop Tests.	87
3.4	Deploy.	88
3.5	Administrate, Configure, and Monitor.	89

1 Why ABAP Cloud?

ABAP Cloud is the ABAP development model to build cloud-ready business apps, services, and extensions.

ABAP was first designed for developing enterprise business applications. Over time, new concepts, technologies, and statements have been added, but not all are built for cloud technology. Therefore, ABAP needed to clearly define which technologies and languages can support all necessary cloud features. ABAP Cloud helps streamline this process by providing a stable ecosystem for developing complex enterprise apps involving multiple parties and various layers of extensions.

ABAP Goes Cloud

ABAP Cloud provides everything needed to develop lifecycle-stable and cloud-ready business apps, services, and extensions. It is scalable and flexible. With ABAP Cloud, you can build apps that can quickly adapt to changing business requirements.

The ABAP Cloud development model offers you end-to-end support for your development process by building on the ABAP language and the ABAP ecosystem. And it leverages all the advantages of Cloud development, such as built-in extensibility, lifecycle-stable APIs, and a cloud-optimized ABAP language.

ABAP Cloud Roadmap

For an overview of the ABAP Cloud road map, see [ABAP Cloud Roadmap Information](#).

Product Availability

The ABAP Cloud development model is delivered with the following products:

Availability of ABAP Cloud

Product	Available	Available ABAP Development Models
SAP BTP ABAP environment	Yes	ABAP Cloud
SAP S/4HANA Cloud, public edition	Yes	ABAP Cloud
SAP S/4HANA Cloud, private edition / SAP S/4HANA	Yes, as of release >= 2022 (7.57)	ABAP Cloud / Classic ABAP

[About This Guide \[page 4\]](#)

This guide gives an overview of the ABAP Cloud development model and how to use it to develop apps, services, and extensions.

[How ABAP Cloud Relates To... \[page 6\]](#)

This chapter introduces some basic concepts and terms that are helpful for understanding ABAP Cloud.

[Evolution Towards ABAP Cloud \[page 9\]](#)

This chapter gives you an overview of the evolution of modern ABAP technologies towards ABAP Cloud. ABAP Cloud is the result of continuous innovation in the ABAP platform and the ABAP language.

[Relevant Personas in ABAP Cloud \[page 10\]](#)

This chapter gives you an overview of the personas in ABAP Cloud.

1.1 About This Guide

This guide gives an overview of the ABAP Cloud development model and how to use it to develop apps, services, and extensions.

This guide focuses on the technical concepts behind ABAP Cloud and how to use ABAP Cloud to develop apps, services, and extensions. It gives an overview of the technologies and concepts in ABAP Cloud. The end-to-end development process is described in the corresponding development guides that are linked in each chapter.

Note

Some features and technologies described in this guide are product-specific and their availability depends on your solution. By default, the guide displays all chapters. If you'd like to read a filtered and product-specific version of the guide, you can use the following links:

- [SAP BTP, ABAP environment](#)
- [SAP S/4HANA Cloud](#)
- [SAP S/4HANA / SAP S/4HANA Cloud, private Edition](#)

How to Read This Guide

For a quick overview.

Check out the [ABAP Cloud Development Model \[page 13\]](#) chapter.

This chapter gives you a quick overview of the most important points.



For a detailed overview.

Check out the [ABAP Cloud Development Model \[page 13\]](#) chapter.

This chapter gives you a detailed overview of ABAP Cloud.

For end-to-end use cases.	<p>Check out the End-to-End Development with ABAP Cloud [page 32] chapter and its subchapters.</p> <p>These chapters give you a top-down view of the end-to-end development process.</p>
For the advantages of ABAP Cloud.	<p>Check out the Built-In Qualities [page 15] chapter.</p> <p>This chapter lists all built-in qualities and advantages that come as part of the ABAP Cloud package.</p>
For key concepts behind ABAP Cloud.	<p>Check out the Key Concepts [page 18] chapters.</p> <p>These chapters give you an overview of the technical concepts that accompany the development with ABAP Cloud.</p>
For how we got to where we're.	<p>Check out the Evolution Towards ABAP Cloud [page 9] chapter. This chapter outlines the ABAP journey in the recent years.</p>
For whom ABAP Cloud is intended.	<p>Check out the Relevant Personas in ABAP Cloud [page 10] chapter. This chapter describes who ABAP Cloud is intended for and why.</p>
For how reuse services can help increase your developer efficiency.	<p>Check out the Reuse Services and Libraries [page 29] chapter. This chapter contains details about how the reuse services and libraries can support your development process.</p>
For some technical insight in the ABAP Cloud programming model.	<p>Check out the Conceptual Background: ABAP Cloud Use Cases [page 38] chapter. This chapter gives you an overview of the design-time architecture.</p>
For an overview of available tools.	<p>Check out the Tools Overview [page 30] chapter. This chapter gives you an overview of the different tools for each persona.</p>

Alternatively...

... watch a recorded live-session about ABAP Cloud.	Developer Discussion about ABAP Cloud  .
... read the ABAP Cloud summary blog.	ABAP Cloud Summary Blog  .
... download and read this guide in PDF format.	ABAP Cloud (PDF)

Featured Documentation for End-to-End Development and Tools

This guide is intended as an umbrella that covers all topics concerning the end-to-end development process for ABAP Cloud from a high-level perspective. You can find detailed guidance and information for tools and development in the following documentation:

Guides	Content
ABAP RESTful Application Programming Model	Details about how to develop apps, services, and extensions for transactional use cases.
ABAP Data Models	Details about how to develop data models for ABAP, focusing on ABAP Core Data Services.
ABAP Analytics	Details about how to develop analytical apps.
ABAP Concepts	Details about ABAP concepts and related frameworks.
ABAP Keyword Documentation	Details about the ABAP syntax and semantics of the programming language.
ABAP Development Tools: User Guide	Details about how to use the ABAP development tools for Eclipse.

1.2 How ABAP Cloud Relates To...

This chapter introduces some basic concepts and terms that are helpful for understanding ABAP Cloud.

📘 Note

If you're already familiar with the SAP S/4HANA extensibility concepts, including integration management and the ABAP ecosystem in general, you can skip this chapter and continue with [ABAP Cloud Development Model \[page 13\]](#).

Different SAP Products

ABAP Cloud as a development model supports your end-to-end development. It defines the standard approach for developing ABAP for cloud-ready and upgrade-stable apps and services including the qualities and best practices that enable you to use the newest innovations. ABAP Cloud is offered as part of different products such as SAP S/4HANA Cloud, public edition, or SAP BTP, ABAP environment. In public cloud products, ABAP Cloud is the only available development model for ABAP. In on-premise or private Cloud products, the classic ABAP development model is also available. It's recommended to use the ABAP Cloud development model whenever possible.

ABAP Cloud itself isn't a product, but a holistic development methodology.

For details about the product availability of ABAP Cloud, see [Product Availability. \[page 3\]](#)

Different Extensibility Options

Extensibility enables you to create or extend custom apps and services, whether they're your own, or delivered by SAP. All extensibility options are anchored in the lifecycle stability concepts derived from the

model-driven architecture and the release contract framework. This guarantees a consistent and loosely coupled extensibility approach.

SAP S/4HANA Cloud offers the following extensibility options:

- **Key User Extensibility:** Low-code/no-code extensions created by key user personas, such as adapting the user interface or adding custom fields.
- **Developer Extensibility:** Pro-code development of sophisticated and tightly coupled extensions.
- **Side-by-Side Extensibility:** Pro-code development of sophisticated and loosely coupled extensions with the SAP BTP, ABAP environment where you also develop with the ABAP Cloud development model. For an overview of extensibility options in SAP S/4HANA Cloud, see [Extensibility](#).

In the classic ABAP development model, classic extensibility technologies are also available. However, classic extensions are directly interwoven and tightly coupled with SAP code. That's why it's recommended to use ABAP Cloud as of SAP S/4HANA or SAP S/4HANA Cloud, private edition 2022 SP00, if possible. For more information about extensibility options in the Classic ABAP development model, see [Extensibility](#).

All extensibility options are built on stable public extension points, which remain unaffected and consistent through upgrades and changes. For an overview of the different extensibility options, see [ABAP Cloud and Different Extensibility Options \[page 26\]](#).

The ABAP RESTful Application Programming Model

The ABAP RESTful Application Programming Model (RAP) was designed to improve upon the ABAP Programming Model for SAP Fiori. RAP was designed to incorporate lessons learned from the previous model. RAP enhanced key architectural concepts that are now integral to ABAP Cloud, like separating the business logic implementation and protocol-specific parts. RAP allows you, for example, to expose the same RAP business object for OData V2 and OData V4. This means that the same implementation can be used for multiple business services, which increases efficiency and reusability. This became a best practice for other scenarios, including embedded analytics and integration services. The ABAP Cloud development model incorporated all best practices from RAP, starting with an architecture blueprint based on a model-driven approach.

In ABAP Cloud, you develop transactional apps and services with RAP. With RAP, you develop transactional apps in ABAP Cloud. In ABAP Cloud analytical apps and integration services are also supported. In addition, the ABAP Cloud development model contains all reuse services, the lifecycle and, tools.

Note

RAP is an important part of ABAP Cloud, but ABAP Cloud contains much more.

OLAP and OLTP

ABAP Cloud supports all common development paradigms, including developing transactional (OLTP) and analytical (OLAP) apps and services. ABAP Cloud supports the following main use cases: transactional use cases, analytical use cases, and integration use cases.

The use cases give you a holistic end-to-end and use-case driven perspective on how to effectively use ABAP Cloud to build apps, services, and extensions.

For more information about programming aspects, see:

- [Conceptual Background: ABAP Cloud Use Cases \[page 38\]](#)
- [Transactional Use Cases - Initial Considerations \[page 43\]](#)
- [Analytical Use Cases - Initial Considerations \[page 45\]](#)
- [Integration Use Cases - Initial Considerations \[page 46\]](#)
- [Tools Overview \[page 30\]](#)


The Clean Core Concept

The clean core concept is closely related to the available extensibility options. Essentially, the clean core concept describes a mindset and method to develop future ready ERPs with systems that are as close to standard as possible, while relying on cloud-compliant extensions and integrations. Modification-free extensibility is an integral part of the clean core concept. It allows you to achieve a clear and stable interface between SAP code and custom extensions ensuring safe upgrades. The ABAP Cloud development model makes clean core-compliant extensions and custom developments possible in ABAP.

The 3-Tier-Model

The 3-Tier Model for SAP S/4HANA Cloud, private edition, and on-premise is a concept to manage the coexistence of classic ABAP development and ABAP Cloud in one system. In tier 1, you develop solely with the ABAP Cloud development model and ABAP Cloud rules are enforced with syntax checks. In tier 2, you develop interfaces for non-released SAP objects to mitigate missing released local SAP APIs for dependency management, and potentially later make it easier switch to a released API. In tier 3, you develop with classic ABAP if necessary. This architecture approach allows you to have a clear separation between cloud-ready code and classic development.

Applying the ABAP Cloud rules is also recommended in the tiers 2 and 3, but the rules are only enforced with ABAP Cloud ATC checks.

For more information, refer to the chapter [Extending a new SAP S/4HANA Cloud, private edition or SAP S/4HANA On-Premise System in the Extend SAP S/4HANA in the cloud and on premise with ABAP-based extensions](#)  guide.

ABAP Language Versions

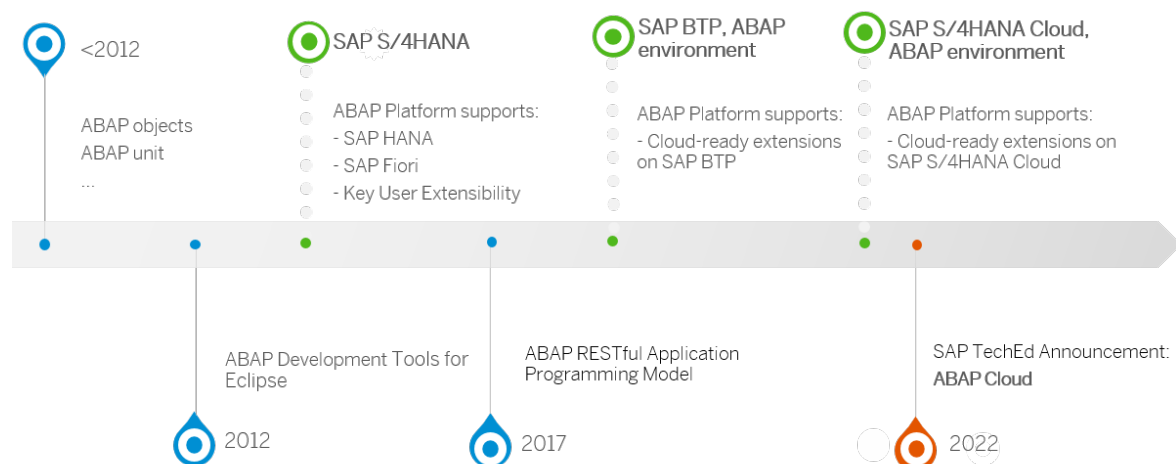
ABAP Cloud builds on a cloud-optimized ABAP language. ABAP language versions determine the available statements, repository objects, and the applicable syntax rules. ABAP Cloud uses the language versions `ABAP for Key Users` in Key User Tools and `ABAP for Cloud Development` in ABAP development tools for Eclipse.

For more information, see [Cloud-Optimized ABAP Language \[page 23\]](#).

1.3 Evolution Towards ABAP Cloud

This chapter gives you an overview of the evolution of modern ABAP technologies towards ABAP Cloud. ABAP Cloud is the result of continuous innovation in the ABAP platform and the ABAP language.

While keeping proven ABAP qualities like integrated lifecycle management, new tools and development concepts were introduced to support customers in their journey towards the cloud:



Evolution of ABAP Development Tools for Eclipse, Core Data Services, and the ABAP RESTful Application Programming Model

The first step was the introduction of the `ABAP Development Tools for Eclipse` as a modern IDE and alternative to the `ABAP Workbench (SE80)`. From a data-model perspective, `Core Data Services (CDS)` were introduced to define and consume semantically rich data models. CDS was then established as the foundation to implement domain-specific transactional, analytical, or integrational use-cases for services and apps.

The `ABAP RESTful Application Programming Model (RAP)` emerged to support the end-to-end development process for transactional services and applications. RAP expanded on the concept of separating the domain-specific implementation from the business service exposure. This architecture blueprint was then adopted as best practice by the other ABAP aspects, for example embedded analytics.

RAP also natively supported many qualities that are the basis for efficient end-to-end and enterprise-scaling development, such as extensibility, documentability, and testability. These integral parts of its development model became the foundation and baseline of the overall ABAP Cloud development model, and they now represent the best practice for all use cases of ABAP Cloud.

Evolution towards the Cloud and Cloud Qualities

For cloud development, upgrade-safety and stability are core paradigms.

To achieve both, and still give customers the possibility to extend SAP applications and development objects, key user extensibility was initially introduced in SAP S/4HANA Cloud as an extensibility option. The release contract framework was scaled and became the foundation for bringing the required cloud qualities together

with the data model developments. This established the foundation for lifecycle-stable extensions with Developer extensibility. With RAP, stateless development for OLTP use cases became the basis for cloud development with ABAP.

All building blocks came together in the ABAP Cloud development model. Combining the strengths of restricted ABAP, CDS, RAP, and modern tooling environments both for developers and key users, the ABAP Cloud development was the result of continuous development around ABAP.

General Availability of ABAP Cloud

With the `SAP BTP, ABAP environment` in 2018, the language version `ABAP for Cloud Development` was introduced so that the ABAP language scope fully supported the cloud paradigms. This enabled developers to leverage ABAP development in a cloud environment.

ABAP Cloud was then introduced to `SAP S/4HANA Cloud, public edition` as part of developer extensibility, enabling developers to develop and extend on-stack in their `SAP S/4HANA Cloud ABAP environment`.

With `SAP S/4HANA 2022 SP00`, ABAP Cloud and developer extensibility are also available as part of `SAP S/4HANA` and `SAP S/4HANA Cloud, private edition` as a development alternative to classic ABAP development. For more information about how extensibility and ABAP Cloud are connected, see [How ABAP Cloud Relates to Different Extensibility Options \[page 6\]](#).

In 2022, ABAP Cloud was introduced as an umbrella term encompassing all ABAP innovations of the last years to define the standard technologies and best practices for modern and cloud-ready ABAP development.

1.4 Relevant Personas in ABAP Cloud

This chapter gives you an overview of the personas in ABAP Cloud.

ABAP Cloud supports administrators, developers, and key users in the end-to-end development process, as shown in the following table:

Personas	Description	Main ABAP Cloud Chapters
Administrators	Administrators are responsible for the configuration, and reliable operation of the ABAP system. The administrator ensures that performance, access to resources, integration setup, and security meet the needs of users. To meet these needs, an administrator monitors the system, maintains security policies and user roles, sets up the integration and automated application jobs, and troubleshoots in case of errors.	<ul style="list-style-type: none">• Integration Use Cases - Initial Considerations [page 46]• Administrate, Configure, and Monitor [page 89]

Personas	Description	Main ABAP Cloud Chapters
Developers	Developers create apps, services, and extensions using the full scope offered in ABAP Cloud to develop apps, services, and extensions for transactional, analytical or integration use cases.	<ul style="list-style-type: none"> • Conceptual Background: Model-Driven Architecture [page 36] • Conceptual Background: ABAP Cloud Use Cases [page 38] • Transactional Services [page 53] • Extend Transactional Services [page 74] • Analytical Services [page 58] • UIs for Analytical and Transactional Apps [page 61] • Develop Integration Services [page 63] • Develop Localization and Internationalization [page 81] • Document Apps and Services [page 82] <p>Analytical Services [page 58]</p> <ul style="list-style-type: none"> • Analytical Use Cases - Initial Considerations [page 45] • Analytical Services [page 58] • Develop Analytical UI Services [page 60] • Extend Analytical UI Services [page 77] • UIs for Analytical and Transactional Apps [page 61]
Key Users, Citizen Developers	Key users can create apps, services, and extensions without advanced development skills using a low code / no code approach. Key users develop with key user tools using the ABAP language version <code>ABAP for Key Users</code> . They can extend released objects with custom fields or additional logic for extensibility use cases or create custom analytical reporting. The functional scope of key users is streamlined to the capabilities offered by the different key user apps.	<p>Key users can find the available option in the product documentation:</p> <ul style="list-style-type: none"> • SAP S/4HANA Cloud: Key User Extensibility • SAP S/4HANA/ SAP S/4HANA Cloud, private edition: Key User Extensibility • SAP BTP, ABAP environment: Extensibility

Personas, Use Cases, and Tools

The following table gives you an overview of the relevant use cases for each persona and which tooling environment the respective persona uses.

Persona	Use Cases	Tools
Administrator	<ul style="list-style-type: none"> Administration Tasks, for example Technical Configuration and Monitoring 	<ul style="list-style-type: none"> SAP Fiori Apps for Monitoring
Key User (Citizen Developer)	<ul style="list-style-type: none"> Develop analytical queries Develop transactional apps 	<ul style="list-style-type: none"> Key User Tools
Developer	<ul style="list-style-type: none"> Develop analytical apps, services and extensions Develop transactional apps, services and extensions Develop system-to-system integration services 	<ul style="list-style-type: none"> ABAP Development Tools

2 ABAP Cloud Development Model

ABAP Cloud is the development model to create lifecycle-stable and cloud-ready business apps, services, and extensions.

ABAP Cloud Overview

The unique strengths of the ABAP ecosystem are integrated into ABAP Cloud. But while ABAP Cloud is firmly integrated into the proven ABAP concepts, such as the lifecycle management or the identity and access management, it also offers other advantages: a release contract framework and a public API lifecycle that guarantees upgrade-safety and lifecycle-stable development.

ABAP Cloud provides tools and techniques that ensure cloud qualities, promotes new technologies, contains a cloud-optimized subset of the ABAP language, and makes upgrade cycles easier by a clear separation between custom code and SAP code by only using released APIs and objects.

The technological core of ABAP Cloud defines the design-time and runtime architecture of all extensions, services, and applications. The main ABAP Cloud elements are:

- ABAP Core Data Services (CDS) for the data models and analytics.
- The ABAP RESTful Application Programming Model (RAP) for transactional apps and services.
- A cloud-optimized ABAP language for the business logic.
- Mandatory public SAP APIs and extension points to allow automated cloud operations and lifecycle-stable extensibility.
- ABAP Development Tools (ADT) as the ABAP IDE.

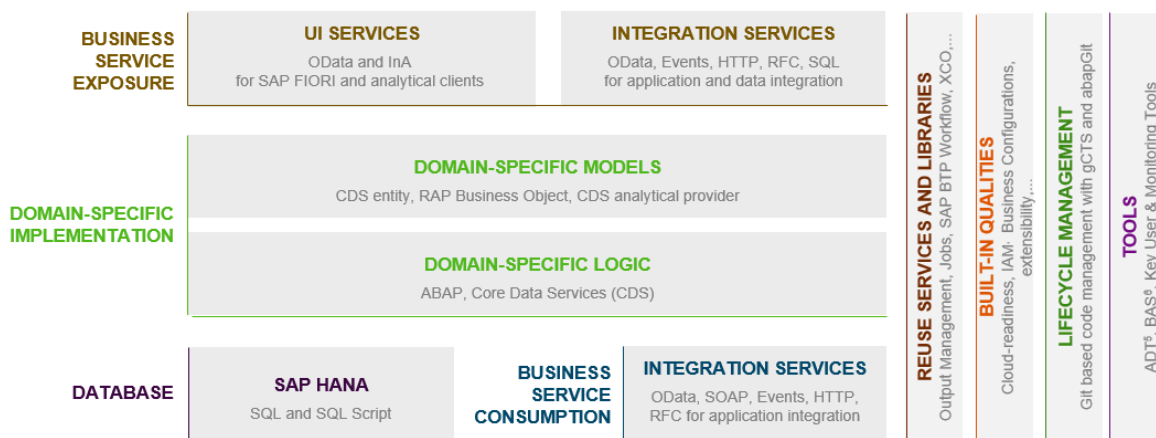
Building on these key elements, you can use ABAP Cloud to cover the following main use cases:

- **Transactional (OLTP):** With ABAP Cloud you can build business objects and expose them as services, to consume them in UIs and integration scenarios. All standard behavior is supported (create, read, update, delete).
- **Analytical (OLAP):** ABAP Cloud is equally equipped for creating services and UIs for data analysis, for drilling down in multiple dimensions, like integrating the data with SAP Analytics Cloud.
- **Integration:** Both previous use cases are complemented by strong data and application integration to cater to today's service-oriented environments.

The development model has two additional two key differentiators:

- The reuse services and libraries with core business services like the number ranges, the application jobs, an ABAP-integrated SAP Fiori Launchpad and UI repository to deploy SAPUI5 and SAP Fiori Elements UIs.
- The built-in qualities offering end-to-end extensibility in the programming model, major cloud qualities like scalability and upgrade stable APIs and many more.

The following graphic illustrates the different building blocks that are part of the ABAP Cloud development model:



- [UIs for Analytical and Transactional Apps \[page 61\]](#)
- [Develop Integration Services \[page 63\]](#)
- [Analytical Use Cases - Initial Considerations \[page 45\]](#)
- [Cloud-Optimized ABAP Language \[page 23\]](#)
- [Analytical Use Cases - Initial Considerations \[page 45\]](#)
- [Reuse Services and Libraries \[page 29\]](#)
- [Built-In Qualities \[page 15\]](#)
- [Deploy \[page 88\]](#)
- [Tools Overview \[page 30\]](#)
- [Develop Integration Services \[page 63\]](#)

Use Cases for ABAP Cloud

With ABAP Cloud, you can cover a variety of use cases for different scenarios. The development path is structured according to the consumers: Either the consumer is a user or system, meaning either you want to create an app to manipulate or display data or you want to exchange data between systems with different protocols to create business processes with system-to-system communication.

With ABAP Cloud, you can develop and extend transactional and analytical apps. Transactional apps have OLTP characteristics to create, update, or delete data records whereas analytical apps only read and display data in charts or dashboards. For more information, see [Develop SAP Fiori Apps \[page 51\]](#) and [Develop Extensions \[page 73\]](#).


Integration services enable the system-to-system communication for different protocols. In ABAP Cloud, integration services can be used for process integration or data integration. With process integration the communication is structured along a predefined business process like, for example, order-to-cash. Data

integration in contrast focuses on transferring raw data without any relation to a business process. For more information, see [Develop Integration Services \[page 63\]](#).

ABAP Cloud in SAP S/4HANA and SAP S/4HANA Cloud, Private Edition

For on-premise and private cloud solutions, ABAP Cloud is embedded in the three tier model concept to enable customers to develop cloud-ready with a clean core strategy. The aim is to separate customer developments and extensions from SAP code as much as possible.

For more details about the three tier model and how it relates to ABAP Cloud, see:

- [How ABAP Cloud Relates to the Clean Core Concept \[page 8\]](#)
- [How ABAP Cloud Completes the 3-Tier Model \[page 8\]](#)
- [Extend SAP S/4HANA in the cloud and on premise with ABAP-based extensions](#)  chapter [Extending a new SAP S/4HANA Cloud Private Edition or SAP S/4HANA On-Premise System](#)

[Built-In Qualities \[page 15\]](#)

This chapter describes the built-in qualities that are shared by all ABAP Cloud use cases.

[Key Concepts \[page 18\]](#)

This chapter provides an overview of fundamental ABAP Cloud concepts.

[Reuse Services and Libraries \[page 29\]](#)

This chapter describes how reuse services and libraries help you save time during the development process.

[Tools Overview \[page 30\]](#)

This chapter gives you an overview of the available tooling environments and when to use them.

2.1 Built-In Qualities

This chapter describes the built-in qualities that are shared by all ABAP Cloud use cases.

Cloud Qualities

ABAP Cloud ensures stability to make sure, that there are no day-1 impacts after and update or upgrade. ABAP Cloud supports scalability regarding dynamic assignment of hardware resources.

Being cloud-ready and scalable with your development processes is a key factor in any cloud environment to ensure automated upgrades. ABAP Cloud supports core cloud qualities that ensure that apps, services, and integrations can run seamlessly in any cloud environment. To achieve this, it's essential that the following cloud qualities are supported by all components of the system, including apps and services.

Separation of Concerns

In classic ABAP all ABAP development objects and language statements can be used. It was possible to modify SAP code so that your extensions and modifications work with the SAP-delivered code. Modifications on

customer side lead to upgrade issues if SAP changes the code that a modification is based upon. There is no clear interface or contract.. With ABAP Cloud, there's a clear interface between SAP code and your custom code, ensuring lifecycle stability and upgrade-safety. This clear separation of concerns is implemented with the release-contract-framework. With this framework SAP releases objects for use in custom custom code.

For more information about how this concept is implemented in ABAP Cloud, see [Public Released APIs \[page 24\]](#).

Controlled SAP LUW

In ABAP applications, the SAP LUW (Logical Unit of Work) concept has addressed the risks of inconsistent data in the database by providing corresponding techniques and data shall be committed to the database with one final COMMIT. However, especially in classic ABAP programming, it was up to developers to make their code comply with the SAP LUW rules without any additional technical checks. Although done unintentionally, it's fairly easy to create implementations that violate LUW rules especially with multiple technical components involved including extensions or even modifications. In ABAP Cloud, the SAP LUW concept is enhanced by the controlled SAP LUW. The controlled SAP LUW introduces a checking mechanism to detect violations of transactional contracts, thus making the SAP LUW more tangible.

The controlled SAP LUW is automatically and implicitly supported by transactional implementation in the context of a RAP Business Object or the local consumption of a RAP business event as well as the background Processing Framework (bgPF). For more information, see [Transactional Consistency \[page 28\]](#).

Model-Driven Architecture

Innovation cycles are shorter than before and force development processes to keep up. The model-based approach, used in ABAP Cloud, ensures a sustainable learning curve that allows you to productively start working with ABAP Cloud based on the key concepts and architecture blueprints. ABAP Cloud ensures a standardized approach to for all use cases so that knowledge can be built up quickly and reused regardless of the specific development use case.

For more information about the model-driven architecture, see [Conceptual Background: Model-Driven Architecture \[page 36\]](#).

Resilience and Scalability

For scalability of an application, it is better to split big processing steps into smaller chunks of work which can be distributed across work processes and, if required, request additional work processes from the infrastructure via load-based auto-scaling. Only if application coding is extracted and executed asynchronously in smaller chunks, then mechanisms like auto-scaling can be applied.

For more resilience some of the processing in a synchronous call from a client should be relocated into asynchronous processing steps, which should be handled in a reliable way by the platform.

Code Quality

ABAP Cloud supports that your code always implements the newest best practices regarding testability, documentability, and supportability. Code quality refers to a set of qualities that help ensure the quality of code during and after development. These qualities include measures for quality assurance, such as support for the test double frameworks, ensuring consistent transactions at runtime, and performing static ATC checks and dynamic runtime checks. In general, all code quality measures ensure that divergence from recommended best

practices is noticed so that developers can always keep the code up-to-date. Documentability ensures that code can be well documented for future maintenance.

Testability

Code testability ensures that a controlled input produces an expected result and shows the expected behavior. ABAP test-double frameworks ensure automated test coverage and the ABAP Test Cockpit helps you maintain and measure the quality of the software.

For more information about how this is integrated in the development process, see [Test \[page 83\]](#).

Documentability

Documentability ensures that all development objects can be adequately documented directly in the development system. Knowledge Transfer Documents (KTD) allow you to document development objects directly on-stack. KTD texts are integrated in the IDE, and are displayed for example in the element info.

For more information, see [Document Apps and Services \[page 82\]](#).

Supportability

Supportability ensures that code can be effectively maintained and supported over time. ABAP Cloud offers a range of tools for developers, which help with the analysis and monitoring of runtime issues. The ABAP Cross Trace allows to troubleshoot issues by providing detailed trace information on how the app logic is executed.

For more information, see [Administrate, Configure, and Monitor \[page 89\]](#).

Extensibility

For cloud lifecycle stability, a clear interface between the different parties is important. Extension points must be well defined and stable. ABAP Cloud enables the extensibility enablement and the building of extensions with ABAP.

For more information, see [Extensibility \[page 26\]](#).

Business Configuration

Business Configuration includes the development, maintenance, and delivery of objects and data that is the basis for business processes or standard code lists, such as, for example, ISO standards.

For more information, see [Conceptual Background: Business Configuration \[page 40\]](#).

Localization and Internationalization

Localization and Internationalization ensure that developments can be adapted to local market requirements, such as, for example, with country/region-specific business configurations or localized labels. In ABAP Cloud, you can use dedicated apps and frameworks to adapt your apps and services.

For more information about how this is integrated in ABAP Cloud, see [Develop Localization and Internationalization \[page 81\]](#).

2.2 Key Concepts

This chapter provides an overview of fundamental ABAP Cloud concepts.

The following chapters explain some of the key concepts that ABAP Cloud is based on, ranging from the supported use cases and front-end technologies to the back-end architecture concept behind ABAP Cloud.

The ABAP Cloud development model builds on the following key concepts to support lifecycle-stable, cloud-ready, and efficient development.

[Broad Use Case Coverage \[page 18\]](#)

The defined use cases assist in defining the possibilities for developing with ABAP Cloud. They provide a path that guides the development of your applications and services in ABAP Cloud.

[Cloud-Optimized ABAP Language \[page 23\]](#)

Different ABAP language versions support the development of cloud-ready apps and services.

[Public Released APIs \[page 24\]](#)

Publicly available APIs, released either locally or remotely, serve as the interface between SAP code and repository objects.

[Extensibility \[page 26\]](#)

One of the differentiating qualities of domain-specific data models in ABAP Cloud is an upgrade-safe extensibility across all programming aspects. This extensibility approach, with its dedicated tools, is the basis for different extensibility options that are offered as part of different SAP products.

[Transactional Consistency \[page 28\]](#)

Transactional consistency guarantees that a database system maintains the integrity and correctness of data during a transaction.

2.2.1 Broad Use Case Coverage

The defined use cases assist in defining the possibilities for developing with ABAP Cloud. They provide a path that guides the development of your applications and services in ABAP Cloud.

All use cases in ABAP Cloud are supported end-to-end. They benefit from the built-in qualities, as well as the use of reusable services and libraries, along with a specialized set of tools for the different use cases and personas.

Table Captions

- **Name:** Name of the programming model aspect as it is used in this guide.
- **Characteristics:** Description of the main characteristics for each programming model aspect.
- **Based On:** Lists the main technologies that are used for domain-specific modeling in the programming model aspect.

- **Implemented With:** Lists the main programming languages that are used to implement the domain-specific logic in each programming model aspect.
- **Implemented For:** Explains which user type an app is intended for. This is important for the authorization considerations in the **Design** phase because the required steps for business roles and authorizations depend on the user type.
 - Business User: Business users are customer-owned end users who use SAP Fiori applications, as well as applications and services in SAP BTP, such as SAP Business Application Studio. These users are authorized by being assigned a specific business role
 - Communication User: Communication users are customer-owned technical users who are assigned to a communication system to enable integration with other solutions.

For details about the user types, see [User Types](#).

📘 Note

This table has invisible columns. You can add them with the [Hide/Show Columns](#) option above the table.

Use Case	Characteristics	Based On	Implemented With	Implemented for
Transactional Use Cases (OLTP)	<p>Transactional use cases require apps and services that run operations such as read, create, update, or delete operations on the data set. The architectural separation of concerns between the domain-specific implementation and the business service exposure lets you expose the same data model for analytical and transactional apps and services.</p> <p>The center piece of RAP is the Business Object (RAP BO). RAP BOs consist of the data model implemented with CDS and enriched with transactional behavior using the behavior definition (BDEF) that is implemented with the <code>Entity Manipulation Language</code>.</p> <p>ABAP Cloud enables you to either develop OData-based services for apps for users that contain UI-specific information, or to develop OData-based Web APIs for unspecified clients, or both at the same time. UI scenarios, namely for creating SAP Fiori UIs, are supported using the OData protocol, while integration sce-</p>	<ul style="list-style-type: none"> • ABAP RESTful Application Programming Model • Core Data Services 	<ul style="list-style-type: none"> • ABAP/Entity Manipulation Language 	<ul style="list-style-type: none"> • For Apps: Business User • For OData APIs and Remote Events: Communication User

Use Case	Characteristics	Based On	Implemented With	Implemented for
	narios are supported with business events, and OData services.			
Analytical Use Cases (OLAP)	<p>Analytical use cases involve analyzing and evaluating multidimensional data models to derive real-time data-driven business decisions. The analytical programming model aspect focuses on creating data models to analyze business data in embedded or cross-system setups and to visualize the data in dashboards or as part of apps.</p> <p>Analytical data models are CDS-based. The analytical provider consists of a reusable star or snowflake schema (based on cubes, dimensions, and hierarchies) and scenario-specific analytical projections (analytical queries).</p> <p>ABAP Cloud enables you to develop InA-based services for multidimensional user apps. The InA services are either consumed in SAP Fiori UIs or by SAP Analytics Cloud.</p>	<ul style="list-style-type: none"> Core Data Services 	<ul style="list-style-type: none"> Core Data Services 	<ul style="list-style-type: none"> Business User

Use Case	Characteristics	Based On	Implemented With	Implemented for
Integration Use Cases	<p>Integration use cases require the integration of different apps and services to work seamlessly in an end-to-end business process. You can use data integration to exchange data between two or more parties without being part of specific and predefined business process, for example, for analytical use cases.</p> <p>Process integration requirements can range from data exchange across system boundaries, such as, for example, to trigger follow-on actions with events when a value in an app is changed.</p> <p>The integration programming model aspect covers all use cases of application-to-application integration across a range of different protocols and frameworks.</p>	<ul style="list-style-type: none"> Core Data Services 	<ul style="list-style-type: none"> ABAP Core Data Services 	<ul style="list-style-type: none"> Communication User

Architecture for the Different Use Cases

Each use case has defines the design time for the specific scenario. The basic architecture for all use cases is based on the RAP architecture blueprint.

For details about the design time architecture and the respective development objects for each aspect, see [Conceptual Background: ABAP Cloud Use Cases \[page 38\]](#).

Tools

There's a dedicated tool set for developers, key users, and enterprise analytics users. For details, see [Tools for Developers \[page 31\]](#) and [Tools for Key Users and Enterprise Analytics Users \[page 31\]](#).

2.2.2 Cloud-Optimized ABAP Language

Different ABAP language versions support the development of cloud-ready apps and services.

About ABAP Language Versions

In ABAP, every repository object has a language version attribute to define the applicable syntax rules and the set of repository objects that can be used as APIs in the implementation.

The language version `Standard ABAP` contains the full scope of ABAP statements and unrestricted access to all repository objects. As a consequence, custom developments and SAP code can be coupled without a well-defined interface in between.

However, this missing interface makes it difficult to predict potential issues after an upgrade. That's why additional ABAP language versions were introduced to minimize the risk of upgrade conflicts and damages to the system or to the data integrity or data security.

`ABAP for Cloud Development` and `ABAP for Key Users` aim to decouple custom developments from SAP code to ensure the upgrade-safety required in cloud development. Only public released SAP APIs (for example released classes and CDS views) are accessible when developing with strict ABAP language versions.

All this means that a strict ABAP Language version consists of a well-defined part of the ABAP syntax of `Standard ABAP` that specifically supports the development process in the cloud for the personas it was created for. Both language versions only allow access to repository objects of the same software component or repository object released with the required release contracts. These stability requirements ensure that your development objects remain stable during upgrades, enabling a seamless upgrade experience.

For more information, see:

- [Released APIs \(ABAP Development Tools: User Guide\)](#)
- [Public Released APIs \[page 24\]](#)
- [ABAP Language Versions \(ABAP Keyword Documentation\)](#)

ABAP Language Versions in Different Product Contexts

In public cloud solutions, ABAP Cloud is the mandatory development model. For SAP S/4HANA and SAP S/4HANA Cloud, private edition, it's recommended to use the ABAP Cloud development model.

Available ABAP Language Versions in Different Products

Products	ABAP for Cloud		Standard ABAP
	Development	ABAP for Key Users	
SAP BTP, ABAP environment	Available	Available	Not available
SAP S/4HANA Cloud, public edition	Available	Available	Not available
SAP S/4HANA / SAP S/4HANA Cloud, private edition (as of 2022)	Available	Available	Available

ABAP Language Versions in ABAP Cloud

ABAP Cloud builds on the cloud-optimized ABAP language versions `ABAP for Key Users` and `ABAP for Cloud Development`.

`ABAP for Key Users` supports key user development with key user tools. `ABAP for Cloud Development` supports the full-stack development of developer and enterprise analytics users with the ABAP development tools for Eclipse. For more information about the respective tooling environments, see [Tools Overview \[page 30\]](#).

Repository objects created with ABAP language version `ABAP for Key Users` and repository objects created with `ABAP for Cloud Development` are separated by name range, software component and/or ABAP packages. `ABAP for Cloud Development` and `ABAP for Key Users` also have separate key user and developer tools for creating, editing, and deleting the respective objects. By default, repository objects can only access other objects with the same ABAP language version and release contract. Repository objects developed with developer extensibility can be released for key user extensibility. Repository objects created with `ABAP for Key Users` can't be released for developer extensibility.

For more information, see:

- [Key User Extensibility and Developer Extensibility](#)
- [Layering of Key User Extensibility and Developer Extensibility \(SAP blog post\)](#) 

2.2.3 Public Released APIs

Publicly available APIs, released either locally or remotely, serve as the interface between SAP code and repository objects.

The basis for upgrade-safety and lifecycle stability in Cloud upgrade cycles is the clear separation of SAP code and customer code, while maintaining stable access to SAP development objects and other functionality for customers. In ABAP Cloud, this is achieved with an API release framework for the different repository objects such as, for example, classes or CDS view entities. With the release contract framework, you can allocate release contracts to different repository objects. The release contract determines the technical purpose of the repository objects, for example if it's intended for use as a local API within your system or as a remote API in

an integration scenario. The visibility of the API state of a repository object determines in which development environment and language version the release contract is applicable.

Depending on the release contract, the respective development objects undergo additional compatibility and consistency checks that ensure lifecycle-stability for the development object consumers. For more information about how to set-up checks, see [Creating API Snapshots](#).

They also impose general restrictions on the released development object to avoid incompatible changes in the future. Once a repository object is released with a release contract, you can safely use it in your implementation. Public released APIs, follow a predefined lifecycle and every deprecated API has a defined successor. Developers using released APIs must consider the allowed compatible changes and access the APIs in such a way that errors and interruptions are avoided. For more information about applicable rules, see [Contract Rules for ABAP Released APIs \(ABAP Keyword Documentation\)](#).

You can find an overview of remote and local released APIs on the [SAP Business Accelerator Hub](#) .

About Release Contracts

- **Extend (C0):** The `Extend` release contract, with the visibility `Use in Cloud Development` or `Use in Key User Apps`, is used for extensibility use cases. Use this release contract to enable, for example, a business object for extensibility.
For more information, see [Extend \(C0\)](#).
- **Use System-Internally (C1):** The `Use System-Internally (C1)` release contract, with the visibility `Use in Cloud Development` or `Use in Key User Apps`, is required for all development objects that you want to use in the domain-specific implementation within your system across different software components.
For more information, see [Use System-Internally \(C1\)](#).
- **Use as Remote API (C2):** The `Use as Remote API (C2)` release contract is used for development objects you want to use in an integration scenario or in a side-by-side extension scenario. For more information, see [Use as Remote API \(C2\)](#).
- **Manage Configuration Content (C3):** The `Manage Configuration Content (C3)` release contract is intended for keeping business configuration content stable. For more information, see [Manage Configuration Content \(C3\)](#).
- **Use in ABAP-Managed Database Procedures (C4):** The `Use in ABAP-Managed Database Procedures (C4)` release contract is used to keep development objects, such as ABAP classes or ABAP interfaces used in ABAP-managed database procedures stable.
For more information, see [Use in ABAP-Managed Database Procedures \(C4\)](#).

Solution-Specific Local and Remote APIs

All local and remote APIs released by SAP in the different solutions follow the release contract approach. Consequently, all released APIs have a well-defined and stable transactional behavior to allow consistent and upgrade-stable usage in all ABAP Cloud programming model aspects. Generally, released CDS view entities follow the virtual data model recommendations, best practices, and naming patterns.

The availability of local released APIs depends on your solution scope. To find all released APIs in your solution, see [Finding Released APIs and Deprecated Objects](#).

2.2.4 Extensibility

One of the differentiating qualities of domain-specific data models in ABAP Cloud is an upgrade-safe extensibility across all programming aspects. This extensibility approach, with its dedicated tools, is the basis for different extensibility options that are offered as part of different SAP products.

Extensibility is a core component of ABAP Cloud. Seamless extensions and stable interfaces between SAP code and custom code is a technical prerequisite for cloud-ready development and upgrade-safety for custom developments. This modification-free approach is supported with release contracts that guarantee the stability of extensions across multiple software components. This is a layered extensibility approach in which multiple parties, such as customers and partners, can extend applications and services in parallel without technical interference.

For a holistic overview of ABAP-based extensions, see [Extend SAP S/4HANA in the cloud and on premise with ABAP-based extensions](#) .

The following explanations focus on the extensibility aspects that are relevant in the context of ABAP Cloud.

ABAP Cloud and Different Extensibility Options

Extensibility as a built-in quality in ABAP Cloud is the basis for different extensibility options offered in SAP solutions. These options include extending released SAP applications, services, and UIs, as well as developing new custom applications and services in the respective systems. ABAP Cloud supports both use cases, including targeted tooling for each extensibility option.

SAP S/4HANA Cloud, SAP S/4HANA, and SAP S/4 HANA Cloud Private Edition offer the following extensibility options based on ABAP Cloud:

	Key User Extensibility	Developer Extensibility	Side-by-Side Extensibility
Scenario	Smaller low/no-code extensions	Tightly coupled, more complex extensions and apps	Loosely coupled extensions and apps
Extensibility Type	On-Stack Extensibility	On-Stack Extensibility	Side-by-Side Extensibility
Target environment	SAP S/4HANA Cloud, SAP S/4HANA, SAP S/4HANA Cloud private Edition		SAP BTP, including SAPBTP, ABAP environment

	Key User Extensibility	Developer Extensibility	Side-by-Side Extensibility
Use cases	<ul style="list-style-type: none"> • UI field layout such as small changes to existing apps (adding new fields, for example) • Custom fields to extend UIs and custom logic to extend business logic • Custom CDS views to create custom data models or custom queries 	<ul style="list-style-type: none"> • Lifecycle-stable, ABAP-based custom app development • ABAP-based extensions of SAP S/4HANA solutions • Lifecycle-stable partner extensions 	<ul style="list-style-type: none"> • Custom or multitenant applications • ABAP and non-ABAP (Java, Node.js) development • SaaS Solutions developed by partners. • Apps for separate target groups
Persona	Key users	Developers	Developers
Extensibility Tools	Key User Extensibility Tools	ABAP Development Tools	ABAP Development Tools
ABAP Language Version	ABAP for Key Users	ABAP for Cloud Development	ABAP for Cloud Development
Benefits	<ul style="list-style-type: none"> • Fully managed and integrated in solutions • No development skills required 	<ul style="list-style-type: none"> • Lifecycle-stable custom ABAP development • Use and extend released SAP objects. • Higher developer productivity • Rich set of stable extension points • No remote access and data replication 	<ul style="list-style-type: none"> • Decoupled extensions independent of solution operation and lifecycle management

Extensibility for Apps and Services for Developers

All use cases offer upgrade-safe extensibility as a built-in quality. The following chapters focus on how to develop on-stack extensibility using the ABAP development tools for Eclipse.

For more information, see:

- [Extend Transactional Services \[page 74\]](#)
- [Extend Analytical UI Services \[page 77\]](#)

For relevant Tools, see:

- [Tools for Developers \[page 31\]](#)

Extensibility for Apps and Services for Key Users

The ABAP Cloud guide focuses on on-stack extensibility with the ABAP development tools for Eclipse. But key users also have multiple extensibility options available to them, such as adding custom fields to services and many other options.

You can find the available option in the product documentation. For more information, see:

- **SAP S/4HANA Cloud:** [Key User Extensibility](#)
- **SAP S/4HANA/ SAP S/4HANA Cloud, private edition:** [Key User Extensibility](#)
- **SAP BTP, ABAP environment:** [Extensibility](#)

For Tools, see:

- [Tools for Key Users \[page 31\]](#)

2.2.5 Transactional Consistency

Transactional consistency guarantees that a database system maintains the integrity and correctness of data during a transaction.

Connection Between Model-Driven Architecture and Transactional Consistency

A model-driven architecture approach like in ABAP Cloud is a design approach where the architecture is built based on a pre-defined conceptual model that defines the structure, behavior, and relationships of the system's components. Transactional consistency can be achieved with a well-defined model, that includes the rules and constraints that ensure transactional consistency. By adhering to this model, developers can ensure that the system behaves consistently and reliably. In ABAP Cloud, this concept is implemented using the controlled SAP LUW in all transactional aspects.

Controlled SAP LUW

In business applications, a transaction describes a sequence of related and/or interdependent actions, such as retrieving or modifying data. When an application runs, data is often modified and is temporarily in an inconsistent state. The result of the transaction, which is considered a logical unit of work (LUW), is a consistent state of the data. An LUW takes an all-or-nothing approach: It ends either with a single and final commit, which persists the changed data in the database. Alternatively, it ends with a rollback (for example, in the case of an error during the LUW), which undoes all changes. The rollback restores the consistent state previous to the changes.

The controlled SAP LUW is an extension of the SAP LUW concept that helps developers address the risks of having inconsistent data in the database. It introduces a checking mechanism to detect violations in the modify

and save transactional phases. This makes applications more robust and avoids transactional inconsistencies. Such a violation can be caused by using invalid operations in the transactional phases modify and save.

For example, using the controlled SAP LUW, database modifications are only allowed in the save transactional phase, not in the modify phase. In addition, APIs (such as methods) can have special classifications that define specific transactional contracts. The classifications will control that the functionality encapsulated by the APIs is used only in specific contexts where it's allowed. This affects both a consumer and a provider of such an API. A consumer isn't allowed to use classified APIs in certain contexts. Similarly, a provider isn't allowed to use certain operations in its implementations.

For more information, see [Controlled SAP LUW](#).

2.3 Reuse Services and Libraries

This chapter describes how reuse services and libraries help you save time during the development process.

A key element of ABAP Cloud are reuse services and libraries. They're offered directly on-stack and automatically come with the respective product environment without any additional cost. Reuse services and libraries offer functionalities to support domain-specific implementation and reduce boilerplate code in your development process.

Reuse services and libraries are well integrated in the ABAP Cloud programming model to optimally reduce the total cost of development for apps and services. All ABAP Cloud reuse services and libraries preserve ABAP Cloud built-in qualities and can be used across all use cases.

2.3.1 Reuse Services

Reuse services offer standard functionality and allow you to significantly increase the developer efficiency.

About Reuse Services

Reuse services aren't application or business object-specific, but they offer general capabilities that are required by multiple services, apps, and business areas. A variety of reuse services are available and ready-to-run, ranging from application jobs that handle logging, forms, change documents, the workflow, and many more.

Released Reuse Services as part of ABAP Cloud

A selected number of reuse services are released as part of ABAP Cloud and are available across all solutions. This means that they offer released APIs to be called at application runtime to use the service, and that configuration and modeling capabilities are offered to configure the service for the specific application.

If required, the reuse services that are released with ABAP Cloud come with a meaningful set of default content, such that the service is ready-to-use in any customer ABAP deployment by default.

UI Reuse Components

For some reuse services, Fiori Reuse Components are provided and can be embedded in Fiori Elements applications. These UI reuse components complement the backend integration in your domain-specific implementation, with a ready-to-run frontend artefact. Two examples are the change document or application log reuse components, which can be embedded in your frontend with little effort.

2.3.2 Libraries

Libraries provide out-of-the-box functionality for common development tasks.

About Libraries

Programming libraries provide reliable, tested functionalities for common tasks, which eliminate the need to write code from scratch. By using libraries, you can promote code readability and maintainability, as they standardize certain processes across different projects.

ABAP Language Library

The ABAP language offers classes for a broad number of tasks such as, for example, technical functions for string processing, parallelization, date and time, as well as runtime type information and XML and XSLT handling.

Extension Components Library (XCO)

The XCO ("Extension Components") library is a general-purpose development library for ABAP that provides an efficient ABAP development experience. XCO consists of the following:

- XCO ABAP Repository Library: Contains highly standardized APIs for the reading and generating ABAP Cloud workbench objects.
- XCO Standard Library: Offers easy-to-consume functions for everyday programming tasks related to JSON handling, XLSX, regular expressions, and other tasks.
- XCO I18N Library: Allows to programmatically maintain translations for language-dependent texts of repository objects.

For more information about the core principles and how to use XCO, see [XCO Library](#).

2.4 Tools Overview

This chapter gives you an overview of the available tooling environments and when to use them.

Each persona in ABAP Cloud has their dedicated tooling environment that is adapted to their specific requirements. For an overview of the personas in ABAP Cloud, see [Relevant Personas in ABAP Cloud \[page 10\]](#).

Tools for Developers

ABAP Development Tools for Eclipse

Developers require a full-fledged integrated development environment that supports the end-to-end development process and to increase developer productivity as much as possible.

The ABAP development tools for Eclipse (ADT) are the integrated development environment on the well-known Eclipse platform for all standard ABAP development, quality assurance, and supportability tasks in ABAP Cloud. ADT offers a modern development toolset with many advantages including full support for development for the domain-specific implementation with ABAP Core Data Services (CDS), ABAP-Managed Database Procedures (AMDP), and the ABAP RESTful Application Programming Model (RAP), or ABAP Analytics.

ADT also offers a range of tools to help with the analysis and monitoring of runtime issues. For details about the tools that are used for monitoring and support, see [Administrate, Configure, and Monitor \[page 89\]](#).

Tools for Key Users and Enterprise Analytics Users

Key users are the business department experts or implementation partner experts who create so-called last mile adaptations with high product skills but with limited technical/coding skills.

Key users use key user tools. Key user tools are web-based Fiori apps that don't require any configuration of the development environment. They provide easy, guided extensibility that's suitable for key users, based on a no code/low code approach. With key user tools, you can implement, for example, adaptations for analytics, forms UI adaptations, custom fields, or logic extensions.

Tools for Administrators

Administrators have technical tasks related to setting up the system. Administrators take care to set up the monitoring or communication management in ABAP Cloud.

ABAP Cloud provides a dedicated tooling set for administrators that allows them to accomplish their tasks. The Fiori apps that are available for administrators include apps for the following:

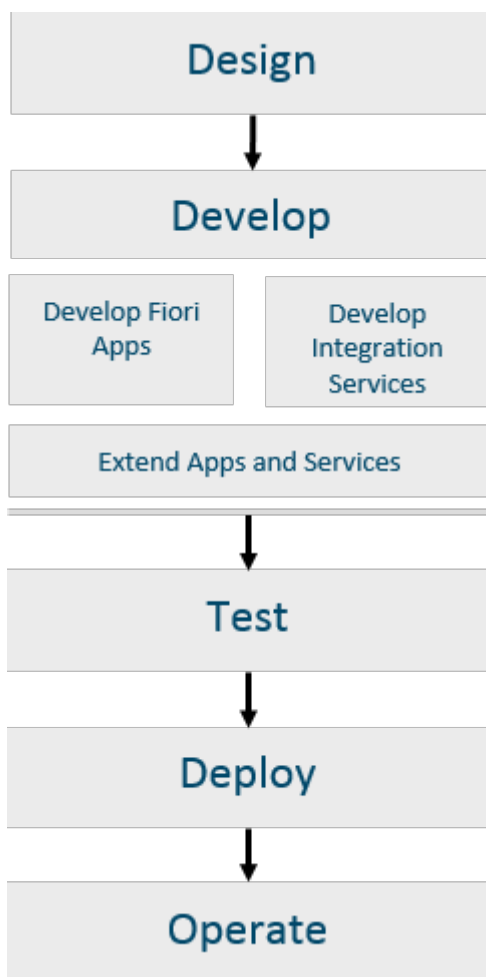
- Identity and Access Management (IAM): IAM apps secure access to the solution for business users.
- Communication management: Communication management is the basis for integration services to enable data exchange across system boundaries.
- Security: Security apps ensure a safe and GDPR-compliant system setup.
- Technical monitoring: With apps for technical monitoring, administrators can monitor the system workload and resource consumption in the ABAP and HANA systems, ABAP work processes, and the performance impact of SQL statements.

3 End-to-End Development with ABAP Cloud

This chapter explains how to put the ABAP Cloud concepts into practice and how to develop apps or integration services end-to-end with ABAP Cloud.

Content Overview

This chapter provides an end-to-end understanding of the ABAP Cloud development process, structured along the different phases of the development lifecycle. Each chapter highlights the most important aspects of the respective phase and the recommended considerations about what to take into account and where to find more detailed information about all mentioned topics.



- [#unique_8/unique_8_Connect_42_design \[page 33\]](#)
- [#unique_8/unique_8_Connect_42_develop \[page 33\]](#)
- [#unique_8/unique_8_Connect_42_fiori_apps \[page 34\]](#)
- [#unique_8/unique_8_Connect_42_integration_services \[page 33\]](#)
- [#unique_8/unique_8_Connect_42_extend \[page 34\]](#)
- [#unique_8/unique_8_Connect_42_test \[page 34\]](#)
- [#unique_8/unique_8_Connect_42_deploy \[page 34\]](#)
- [#unique_8/unique_8_Connect_42_operate \[page 34\]](#)

This graphic gives you an overview of the content of the different chapters. Click on each development phase for more information.

Design

The design chapters give you an overview of the design principles behind ABAP Cloud, the main use cases, and the authorization basics you should take into account before starting to develop an app or an integration service.

Design

- [Conceptual Background: Model-Driven Architecture \[page 36\]](#)
- [Conceptual Background: ABAP Cloud Use Cases \[page 38\]](#)
- [Transactional Use Cases - Initial Considerations \[page 43\]](#)
- [Analytical Use Cases - Initial Considerations \[page 45\]](#)
- [Integration Use Cases - Initial Considerations \[page 46\]](#)

Develop

The develop chapters summarize the development flow for developing an analytical or transactional Fiori app or developing an integration system for system-to-system integration. You can also learn about extensibility for analytical or transactional data models, as well as localization and internationalization.

Develop with ABAP Cloud

- [Develop \[page 49\]](#)
- [Develop SAP Fiori Apps \[page 51\]](#)
- [Develop Extensions \[page 73\]](#)

Develop Integration Services

The integration services chapter summarizes the possibilities for developing data and process integration using different protocols or event-based communication.

Develop Integration Services

- [Develop Integration Services \[page 63\]](#)
- [Consume External Integration Services \(Outbound Communication\) \[page 70\]](#)
- [Expose Integration Services \(Inbound Communication\) \[page 69\]](#)
- [Develop Event-Based Integration \[page 72\]](#)

Develop Fiori Apps

The Fiori apps chapters summarize how to develop a transactional or analytical app with ABAP Cloud.

Develop Analytical or Transactional Fiori Apps.

- [Develop Transactional UI Services \[page 55\]](#)
- [Develop Transactional Services for Business Configuration \[page 57\]](#)
- [Develop Analytical UI Services \[page 60\]](#)
- [UIs for Analytical and Transactional Apps \[page 61\]](#)

Extend Apps and Services

The extend chapters summarize how you can extend apps and services with built-in extensibility for CDS data models and RAP business objects. The ABAP Cloud chapters focus on on-stack extensibility that's developed with the ABAP development tools for eclipse. For extensibility options for key users, see [Extensibility for Apps and Services for Key Users \[page 28\]](#).

Extend Apps and Services

- [Extend Transactional Services \[page 74\]](#)
- [Extend Analytical UI Services \[page 77\]](#)

Test

The test chapter summarizes the available test-double frameworks and tools for quality assurance.

Test Apps and Services.

- [Develop Tests \[page 87\]](#)

Deploy

The deploy chapters give you an overview of the product-specific lifecycle management solutions that you can use to transport your development objects.

Deploy your App or Service.

- [Deploy \[page 88\]](#)

Administrate, Configure, and Monitor

The operate chapters give you an overview of what to do when your app or service is actively used in a live environment, such as monitoring or setting up communication scenarios, to enable system-to-system communication.

Administrate, Operate, and Support

- [Administrate, Configure, and Monitor \[page 89\]](#)

[Design \[page 35\]](#)

This chapter explains the requirements and considerations during the design phase for each use case.

[Develop \[page 49\]](#)

This chapter explains the requirements and considerations during the develop phase for each programming model aspect.

[Test \[page 83\]](#)

This chapter explains the requirements and considerations during the test phase.

[Deploy \[page 88\]](#)

This chapter explains how to deploy and transport your app or service with ABAP lifecycle management.

[Administrate, Configure, and Monitor \[page 89\]](#)

This chapter explains the requirements and tools involved with, for example, monitoring an app or service that you've developed.

3.1 Design

This chapter explains the requirements and considerations during the design phase for each use case.

Decide on the Use Case

In ABAP Cloud, programming aspects define the design-time and runtime requirements for the three major use cases: Developing analytical, or transational apps and services, or developing integration services. They give you a use-case driven perspective on the development options that are offered as part of ABAP Cloud and give guidance when developing with ABAP Cloud:

- **Transactional Use Case:** The transactional aspect implements OLTP use cases where CRUD operations are required. The transactional aspect is based on the `ABAP RESTful Application Programming Model`.
- **Analytical Use Case:** The analytical aspect implements OLAP use cases where multi-dimensional data models are queried to analyze business data and service business KPIs.
- **Integration Use Case:** The integration aspect implements system-to-system communication use cases to enable data or process integration between systems.

Since all use cases follow a common architecture blueprint in a model-driven design approach, they build on the same development artifacts from data modeling to consumption. ABAP Core Data Services (CDS) are the basis for all domain-specific data models.

For an overview of the use cases, see [Broad Use Case Coverage \[page 18\]](#).

Get to Know the Basic Design Time Architecture

The design-time architecture for all programming aspects is derived from the transactional programming aspect. All use cases build on a three-tier architecture from **data access**, to **domain-model and implementation** and **business service exposure**. For more details, see [Use Case Architecture - Common Characteristics \[page 38\]](#).

The domain-specific CDS models of programming aspects are multipurpose: For example you can use the same stack for a transactional service and an analytical service by using the respective annotations, and then exposing the data model in as InA service and OData service.

Draft the Authorization Concept

Depending on your use case, you need to define authorizations for business users or communication users to protect your app or service from unauthorized access. For more information, see [Conceptual Background: Access Management \[page 41\]](#).

Additional Considerations

Transactional and analytical apps and services require a UI. ABAP Cloud relies mainly on backend-driven UI development, but also allows breakouts using the `Business Application Studio`. For more information about developing UIs, see [UIs for Analytical and Transactional Apps \[page 61\]](#).

What's Next

Once you've decided on your use case and have checked out the basic design time, you can start to develop your app or service: [Develop \[page 49\]](#).

[Transactional Use Cases - Initial Considerations \[page 43\]](#)

The transactional programming model aspect (transactional aspect for short) defines the OLTP use case for OData-based UI services and integration services, as well as events.

[Analytical Use Cases - Initial Considerations \[page 45\]](#)

Analytical use cases describe the OLAP use cases for InA-based UI services.

[Integration Use Cases - Initial Considerations \[page 46\]](#)

The integration use cases enable services to be exposed and consumed.

3.1.1 Conceptual Background: Navigating the Foundations

3.1.1.1 Conceptual Background: Model-Driven Architecture

ABAP Cloud is based on a model-driven architecture approach that focuses on improving development efficiency through standardization and formalization of the programming model and the tooling environment to ensure efficiency and scalability.

Programming models generally define the design-time software architecture with specific technologies, concepts, and development objects. It essentially defines a standard architecture for app and service development, from the database to the business service exposure.

ABAP Cloud builds on the strengths of powerful frameworks and a standardized architecture for different use cases, to save as much implementation time as possible while providing you with flexibility. You can model your business processes with apps and services based on your business requirements along predefined technical processes. The runtime orchestration is handled by the frameworks whenever possible, to decrease the probability of consistency errors during runtime, especially in implementations that involve multiple business

objects or services. This standardized and consistent architecture across all apps and services developed with ABAP Cloud has many advantages from a development perspective:


- **Efficiency Increase and Scalability:** Developer efficiency is increased, because standard architecture patterns are easily scalable by definition. Once a developer is familiar with developing with the ABAP Cloud development model, the additional effort decreases with each developed service or application. For more information about developer efficiency as built-in quality, see [Built-In Qualities \[page 15\]](#).
- **Adaptability and Maintenance:** A standardized architecture fosters quality code and thus testability and code maintenance. That makes an implementation future-proof - even though new functionality is added over time, the architecture ensures that the model is always adaptable for future changes. ABAP Cloud comes with specific mock-frameworks for data models and events that support the code quality assurance, and it avoids regressions on all test levels. For more information about testability as a built-in quality, see [Built-In Qualities \[page 15\]](#).
- **High-Abstraction Level:** The development model runs all low-level technical and infrastructure-related tasks. This enables you to focus on the domain models and how you want to implement your business process with ABAP Cloud. For more information about the development with ABAP Cloud, see [Develop \[page 49\]](#).

ABAP Cloud was designed with a tightly coupled integration between the domain-specific data model and the domain-specific implementation in mind. The domain-specific cloud-optimized ABAP languages, such as `Data Definition Language` or the `Entity Manipulation Language` match the data modeling requirements and are designed to support the modeling and ABAP-specific development processes as much as possible.

This tight integration between the ABAP world and ABAP Cloud ensures that the development model supports the transition from classic ABAP towards ABAP Cloud while supporting cloud-native characteristics for Cloud environments.

Transactional Consistency Across Apps and Services

The standardized architecture and development approach in ABAP Cloud also ensures the integration capabilities between apps and services in end-to-end business processes. Interoperability is guaranteed because all implementations follow the same technical rule set, and the framework determines the technical contracts and process flows. This enables you to design end-to-end processes without having to worry about how to implement an authorization or locking concept for only for one specific part of the process. Instead, you make process design decisions that incorporate the technological advantages of the different ABAP Cloud technologies, frameworks, and building blocks. You can do this at each step of the process, instead of being limited by technological constraints that are caused by inconsistent design-time and runtime architecture.

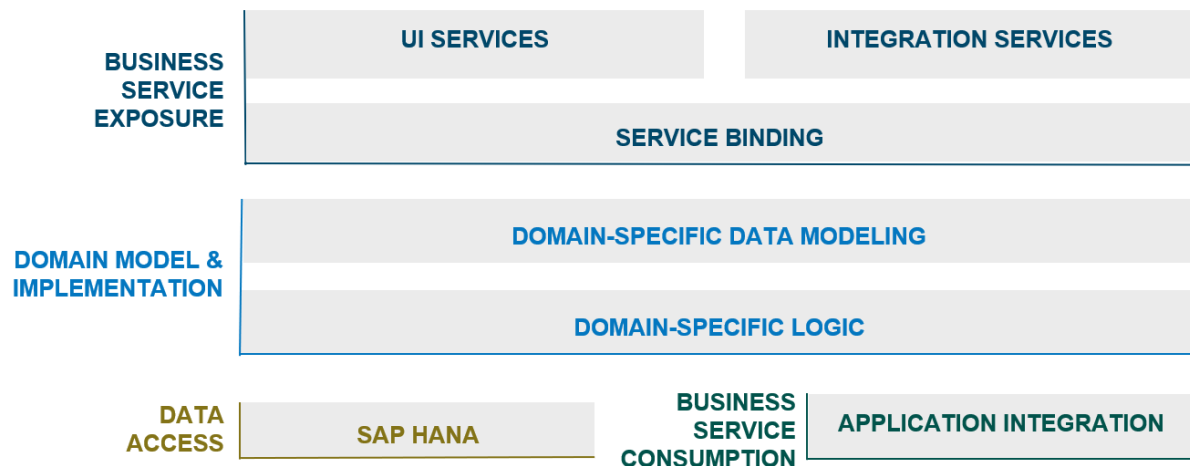
For more information about the transactional consistency in ABAP Cloud, refer to the blog post [SAP LUW in ABAP Cloud](#) .

For more details about transactional consistency as built-in quality, see [Transactional Consistency \[page 28\]](#).

3.1.1.2 Conceptual Background: ABAP Cloud Use Cases

The different use cases define the design-time architecture for ABAP Cloud.

The architecture of all ABAP Cloud use cases shares common design-time characteristics that were derived from the RAP architecture. It served as an architecture blueprint for the use cases. The following graphic illustrates an abstracted design-time architecture that all of them have in common:



- Data Access through database or service consumption model: The data layer for the use cases consists either of an SAP HANA database, where data is queried with SQL and SQLScript, or a service consumption model where the data source originates from a different system.
- Domain-model and implementation: The domain-model and implementation define the application tier of ABAP Cloud consisting of the domain-specific data modeling and the domain-specific logic. The domain-specific modeling defines the data model where the data is read and the domain-specific logic implements the processing of the data, or in case of a transactional app or service, the respective behavior for a RAP business object.
- Business service exposure: Defines an additional abstraction layer between the client and the service exposure. The business service exposure defines the protocol in which a data model and its implementation is exposed for consumption by different clients.

ABAP Cloud follows a three-tier architecture approach, ensuring scalability and separation of concerns between data model, implementation, and the respective service protocol. For more details about the advantages of this approach, see [Conceptual Background: Model-Driven Architecture \[page 36\]](#).

About Domain-Specific Logic

The domain-specific logic is implemented with ABAP, CDS, or a combination of both depending on the specific runtime characteristics of the use cases. The domain-specific logic implemented with CDS is available for all use cases.

Core Data Services

Based on ABAP Core Data Services (CDS), you can define semantically rich data models and make full use of a service model infrastructure that allows you to easily expose your data model. For example, expose your

data model as an OData as an OData service for an SAP Fiori application in the transactional aspect, or as InA service for analytical clients, or an integration scenario. The fully integrated SAP Customer Identity and Access Management supports you in creating and maintaining the role-based authorization management that's required to protect your applications from unauthorized access. CDS data models provide consumers with flexibility by allowing extensions.

Extensibility is fully integrated for all use cases and can be used to enhance your data models with field extensions, or additional behavior for the transactional programming model aspect. Additionally, various CDS tuning capabilities support you in optimizing your data model for better runtime behavior.

For more general information about CDS, see [ABAP Data Models](#).

Cloud-Optimized ABAP

In ABAP, every repository object has a language version attribute to define the applicable syntax rules and the set of repository objects that can be used as APIs in the implementation. Language versions are a central concept for the domain-specific implementation. For details about the ABAP language version concept, see [Cloud-Optimized ABAP Language \[page 23\]](#).

This chapter focuses on the most important language aspects that are relevant for the domain-specific implementation in the different use cases.

About Domain-Specific Data Modeling

Domain-specific data models define the design-time architecture for domain-specific implementations, depending on the characteristics of a programming model aspect. Most parts of the use cases share a common architecture and rely on the same development objects. These development objects, such as Core Data Services (CDS) entities, comprise the RAP business object, the analytical provider, or the CDS data models for the integration aspect like events.

Core Data Services

With CDS, you can define domain-specific data model characteristics that are different for each programming model aspect. Every programming model aspect has its own provider contract, domain-specific annotations, and domain-specific models. They determine the characteristics that are required by the respective runtime frameworks to process the domain-specific data model characteristics, for example, multidimensional data models in analytics. The analytical programming model has specific requirements regarding the data model that is added to the data model with domain-specific additions in CDS.


3.1.1.3 Conceptual Background: Business Configuration

Business Configurations include the development, maintenance, and delivery of objects and data that together form the basis for business processes or standard code lists, such as for example, ISO standards, industry, localization, or standards for best practices.

Predelivered Business Configuration Content

Business configuration content is highly standardized content, such as ISO codes for languages. That's why business configuration content is predelivered with SAP solutions. Your available business configuration content depends on your specific SAP solution.

Public cloud SAP solutions support:

- Content delivery through business configuration sets: Business configuration sets can be delivered as part of a SaaS solution and deployed to a consumer tenant .
- Content delivery through an external content management system, for example [SAP Central Business Configuration](#). For more information, see [Configuring with SAP Central Business Configuration](#) .

Currently, both technologies are used only by SAP development.

Business Configuration in Classic ABAP and ABAP Cloud

Note

For ABAP Cloud in SAP BTP, ABAP environment and SAP S/4HANA Cloud, using the RAP-based business configuration apps is the only available option.

In classic ABAP, you've two options to create business configuration apps: You can create SAP GUI-based business configuration apps with IMG integration or RAP-based configuration apps. You can decide based on your use case which option best suits your requirements.

SAP Fiori/RAP-Based Business Configuration Apps

The RAP-based approach is recommended for green field development use cases or apps with advanced UI requirements and rich custom logic. However, some features aren't yet supported with the RAP-based apps and if these features are required, then you must use SAP GUI-based configuration apps. The following features aren't yet supported:

- Solution Manager Integration for customizing synchronization.
- Cross client comparison (SCUO).
- Content delivery via BC sets for custom configuration tables.

SAP GUI-based Configuration Apps

The SAP GUI-based approach is recommended for use cases that only require a simply UI and limited custom logic. If you already have an SAP GUI-based business configuration app, continuing to use this approach may ensure consistency in the development.

You can transform their maintenance views to SAP Fiori/RAP using the generator for business Configuration objects, if the configuration tables fulfill the prerequisites of the generator. For more information, see [Business Configuration Maintenance Object](#).

Develop Business Configuration Apps

For more information about how to develop business configuration apps, see [Develop Transactional Services for Business Configuration \[page 57\]](#).

3.1.1.4 Conceptual Background: Access Management

This chapter introduces the background concepts and terminology for access management. With access management, you can define and restrict access to apps and services.

About Access Management

Access management ensures that only authorized business and technical users can access specific apps and data.

Note

This chapter gives only a short overview of the important points about identity and access management considerations for SAP S/4HANA Cloud, public edition and SAP BTP, ABAP environment. For a detailed overview of these products, see [Authorization Basics](#).

For the concepts in SAP S/4HANA or SAP S/4HANA Cloud, private edition, see [User and Role Administration of ABAP Platform](#).

User Concepts

Different user concepts are needed to organize authorizations for apps and services. The key distinction lies in who the authorizations apply to. They can apply to business users who are interacting with the system through a UI, or a technical user in the following integration scenario:

- **Business Users:** Users at a customer site who use SAP Fiori apps and log on via the `Identity Authentication Service (IAS)`. Business users can also be used in principal propagation scenarios when the user is already logged on at the sender system and the user context is propagated to the integration partner.
- **Communication Users:** Customer-owned technical users who enable integration with other systems for authentication, using basic authentication or a mapped X.509 client certificate.

Design Time for Access Management

Both the Business User and the Communication User require development objects to group authorizations. In addition, you manage access to data and available operations on the level of individual services. The design time for access management is defined and implemented by the developer persona.

Design Time for Business User Authorizations

Authorizations for business users are modeled in an Identity and Access Management (IAM) app. The IAM app defines the authorizations, for example, for an SAP Fiori app. It groups UI services and therefore provides access to them. Multiple IAM apps for a similar use case are grouped in business catalogs by the developer. The developer can also define business role templates. This makes it easier for administrators to create business roles that are based on relevant business catalogs for the corresponding role in the company.

For more information, see [Providing Access to a Business Service for Business Users](#).

Design Time for Communication Users

Authorizations for communications users are defined with Communication Scenarios. They group integration services and therefore provide access to them.

For more information, see [Providing Access to a Business Service for Communication Users](#).

Design Time for Complex Access Control

Often a more complex access control is required, for example to manage access to data for ABAP Core Data Services (CDS), or to manage who can modify data with transactional services.

With authorization objects, you can restrict the available operations or the available data to authorized communication or business users only. For CDS, you can use access controls in the Data Control Language (DCL) to restrict access to data. For transactional services, you can use explicit ABAP statements that are, for example, checked by the authorization handler of a behavior definition.

You can define default authorization values for each authorization object and service. These values document which authorization objects might be checked at runtime. They are then automatically assigned when the service is added to an IAM app or to a Communication Scenario.

Optionally, it is also possible for most APIs to use the Privileged Mode to suppress authorization checks, for example when authorization checks are not relevant in a certain context.

The list of required authorization objects, with defined activities and field values, can be added to IAM apps and Communication Scenarios. Optionally, it is also possible to delegate the specification of field values for specialized roles to the administrator. These specification options are enabled by creating a generic Business Catalog with Restriction Types and Restriction Fields.

Configuration Time for Access Management

The ABAP access management offers UIs that enable the administrator persona to create and transport access management configuration.

For business user access, the administrator creates business roles to group and fine-tune business catalogs and assign them to business users. Business roles can be transported in a multi-system landscape.

For communication users, the administrator creates communication arrangements, based on a communication scenario, to assign the necessary authorizations.

Basic Considerations for Access Management

For an overview of basic considerations for access management when developing an app or service, see:

- [Transactional Use Cases - Initial Considerations \[page 43\]](#)
- [Analytical Use Cases - Initial Considerations \[page 45\]](#)
- [Integration Use Cases - Initial Considerations \[page 46\]](#)

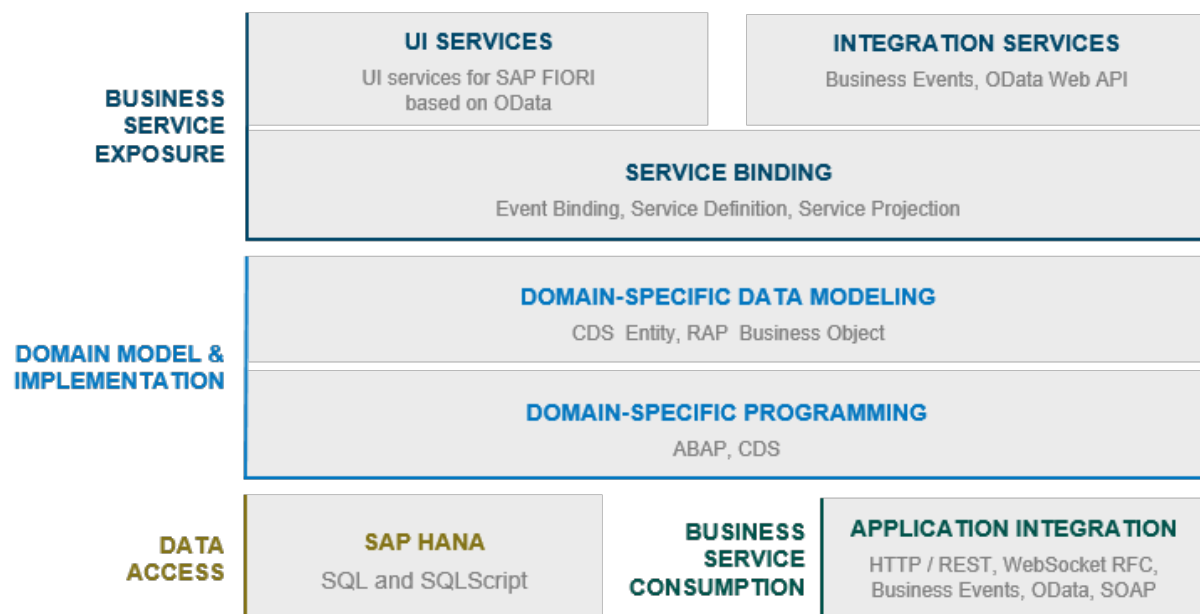
3.1.2 Transactional Use Cases - Initial Considerations

The transactional programming model aspect (transactional aspect for short) defines the OLTP use case for OData-based UI services and integration services, as well as events.

Transactional Design-Time Architecture

OData-based UI and integration services are implemented with the ABAP RESTful Application Programming Model (RAP). You can use RAP to implement UI services to develop SAP Fiori apps or to develop events or Web APIs.

The following graphic gives you an overview of the main parts of the RAP architecture:



- UI services or OData integration services: A RAP business object is at the core of the domain-specific data modeling. The RAP business object defines the behavior for a set of hierarchical CDS entities.

The behavior for the domain-specific logic, such as actions or functions are implemented using ABAP, specifically the `Entity Manipulation Language`. The service projection is defined with the provider contract `transactional_query`.

The business service is then exposed with OData either as a UI service or Web API.

- Business events: The event payload is defined with a CDS abstracts entity and its behavior definition. The event is then exposed using an Event Consumption Model and can be consumed in cross-system setups.

Transactional Runtime

OData requests are handled by SAP Gateway. It provides an open, REST-based interface that offers simple access to SAP systems through the Open Data Protocol (OData).

Incoming requests for RAP business object or the business events are dispatched by the `RAP Runtime Engine`.

For more information, see [Runtime Frameworks](#).

Design-Time for Restriction of Data Access and Activities for Transactional Apps and Services

Note

This paragraph gives only a short overview of relevant considerations for access management for transactional and services. For a holistic overview, see [Authorization Control](#). More details about the conceptual background for access management, see [Conceptual Background: Access Management \[page 41\]](#).

To restrict access to domain-specific logic, such as actions or functions of a business object, the CDS behavior definition offers an authorization handler to check the corresponding authorization objects. In the ABAP code outside the authorization handler, the authorization checks of a called API (for example, CDS entities or other behavior definitions) is not relevant, or even not allowed to maintain transactional consistency. For this reason, all released CDS views and business objects offer a privileged mode to suppress their authorization checks at runtime. When building your own behavior definition, it's good practice to also offer a privileged mode, so that unnecessary authorization checks can be skipped by consumers of your API.

Develop Transactional Apps or Services

With RAP, you can develop apps or integration services that are based on OData. For more information about developing with RAP, see the following topics:

- [Develop Transactional UI Services \[page 55\]](#)
- [UIs for Analytical and Transactional Apps \[page 61\]](#)
- [Develop Integration Services \[page 63\]](#)

- [Develop Tests \[page 87\]](#)

Local and remote business events rely on the same design-time architecture as transactional services, but are used to establish asynchronous communication between apps or services. For more information about business events, see [Develop Event-Based Integration \[page 72\]](#).

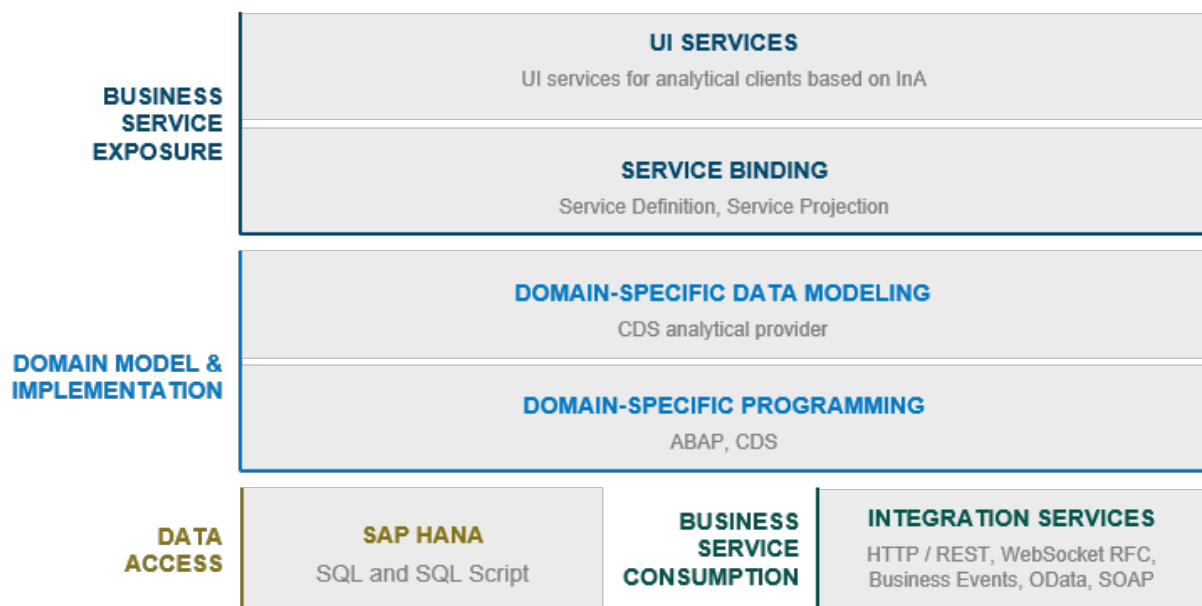
3.1.3 Analytical Use Cases - Initial Considerations

Analytical use cases describe the OLAP use cases for InA-based UI services.

Analytical Design-Time Architecture

Analytical services are implemented with CDS. With CDS, it's possible to define multidimensional data models that are required for analytical queries. CDS can be used to implement UI services for consumption by analytical clients such as, for example, *SAP Analytics Cloud*.

The following graphic gives you an overview of the main parts of the analytical architecture:



- **UI Services:** An analytical provider is at the center of the analytical programming model aspect. The analytical provider defines the multidimensional data model, based on a cube and an analytical query. The domain-specific logic is implemented using ABAP and CDS, for example, to calculate measures with formulas or exception aggregations. The projections service is defined with the provider contract `analytical_query`. The business service is then exposed with the Information Access (InA) protocol for consumption by analytical clients.

Analytical Runtime

InA requests are handled by the ABAP Analytical Engine. The ABAP Analytical Engine processes multidimensional data models and performs measure calculations either on the SAP HANA database or in ABAP.

Design Time for Restriction of Data Access for Analytical Apps

Note

This paragraph gives only a short overview of relevant considerations for access management for transactional and services. For a holistic overview, see [Identity and Access Management](#). More details about the conceptual background for access management, see [Conceptual Background: Access Management \[page 41\]](#).

To restrict read access to data in a multidimensional data model, it's sufficient to use data control language (DCL) for the CDS entities used in the data model. The CDS runtime uses conditions defined in CDS access control to limit the result set returned by a CDS entity, often by checking the corresponding authorization objects.

For more information about DCL, see [CDS Authorization Concept](#).

Develop Analytical Apps and Services

You can develop multidimensional data models and develop evaluations and reports and KPIs in real-time, based on up-to-date business data with embedded analytics, or choose a side-by-side setup.

For more information about developing embedded analytics, see these topics:

- [Analytical Services \[page 58\]](#)
- [UIs for Analytical and Transactional Apps \[page 61\]](#)
- [Develop Tests \[page 87\]](#)

3.1.4 Integration Use Cases - Initial Considerations

The integration use cases enable services to be exposed and consumed.

The integration services cover both point-to-point integrations and integrations using event-driven architectures. ABAP Cloud enables integration between SAP cloud products, SAP BTP services, on-premise landscapes, customer extensions, and external services. This chapter covers both the design-time aspects for the developer persona and the configuration-time aspects for the administrator persona.

The exposure of services to other external systems is referred to as inbound communication. The consumption of services from external systems is referred to as outbound communication.

For an overview of the different personas, see [Relevant Personas in ABAP Cloud](#) [page 10].

📌 Note

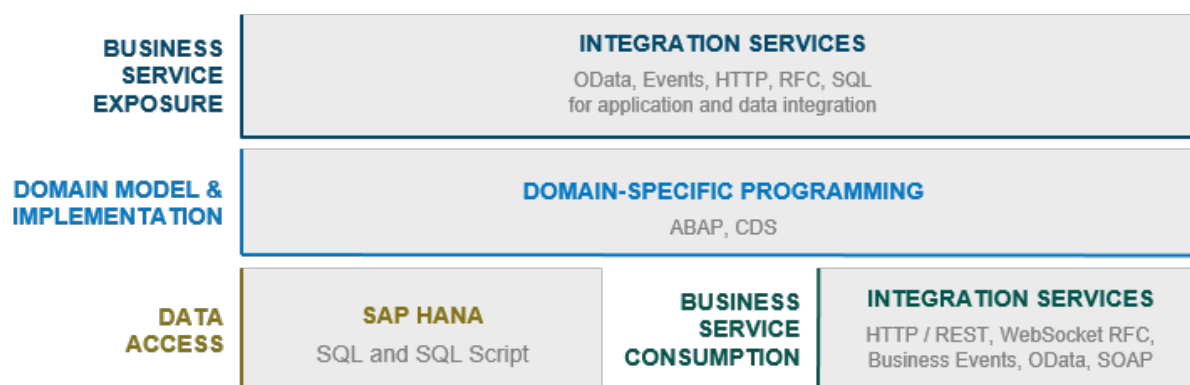
This chapter only gives a short overview of important points for the integration programming model aspect for SAP S/4HANA Cloud and SAP BTP, ABAP environment. For the concepts in SAP S/4HANA and SAP S/4HANA Cloud, private edition, see:

- [Developing External Service Consumption \(Outbound Communication\)](#)
- [Developing APIs for Inbound Communication](#)

Integration Design-Time Architecture

ABAP Cloud offers various possibilities to integrate with other systems using OData, SOAP, HTTP, RFC, and SQL protocol. In addition, you can use the SAP Event Mesh to expose and consume business events.

The following graphic gives you an overview of the main parts of the integration design-time:

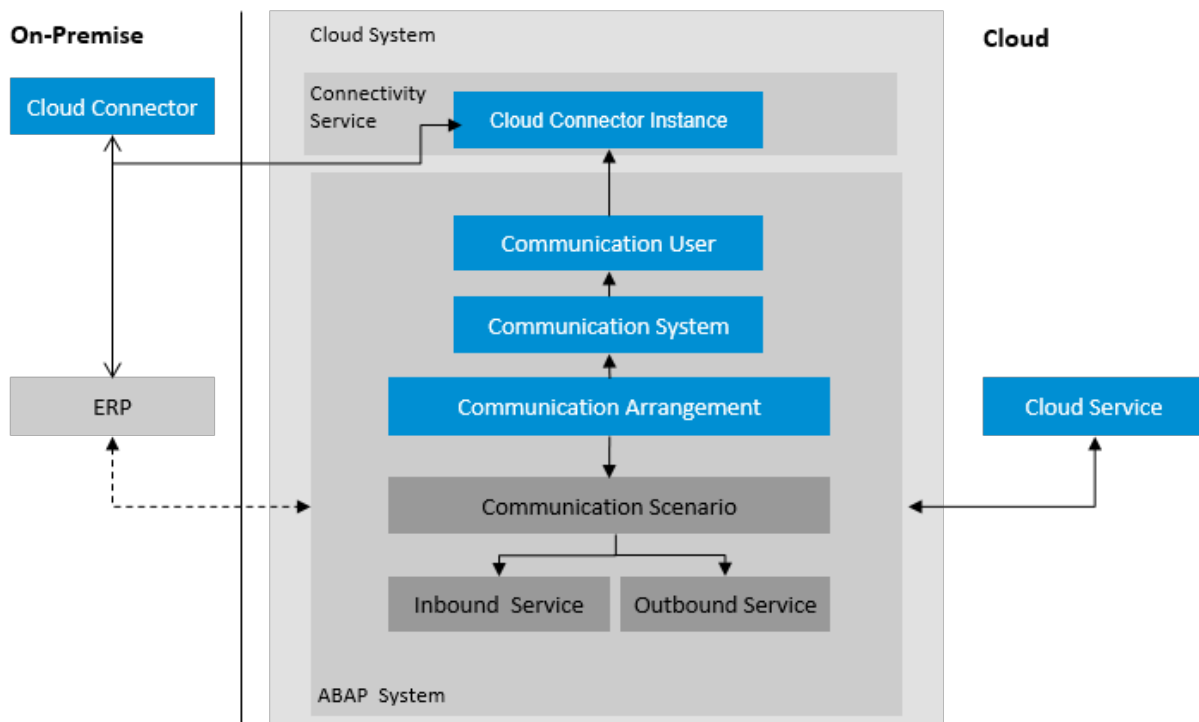


- **Integration Services Exposure:** The development model for exposing integration services depends on the respective protocol. OData, SQL, and business events interfaces can be created using the architecture and development objects of the RAP programming model. The domain-specific logic of SOAP, HTTP, and RFC services is implemented with ABAP.
- **Integration Service Consumption:** The consumption of services and events from other systems is simplified by using metadata to generate an Event Consumption Model and a Service Consumption Model. Service Consumptions Models are available for OData, SOAP, RFC, and business events. For HTTP and RFC, you can define the consumption requests directly in ABAP.

With communication scenario as a development object, you can bundle inbound and outbound integration services to simplify the setup process for the administrator. The communication scenario defines the protocols that are used for the integration.

Integration Configuration Time

Communication management involves tasks that are divided between administrators and developers. The following graphic illustrates the various components of communication management and how they relate to each other:



Using the communication scenarios as design-time artifacts, the administrator of the system can set up technical integrations by using the `Communication Management` apps. A communication arrangement instantiates a communication scenario by assigning a communication system to a communication scenario. This creates a configured and executable instance of the scenario in the respective system. The communication system represents an integration partner, with a concretely assigned communication user for inbound integrations, or it sets up outbound integration. For outbound communication, the administrator maintains the system host name and the credentials of the integration partner. Various standard-based authentication methods are supported.

In hybrid landscapes, the `SAP Cloud Connector` is used to integrate with protected internal customer landscapes by providing detailed control through a secure tunnel connection. To distribute business events, the administrator can connect the system to an SAP Event Mesh instance and configure which event topics should be published to it.

Integration Runtime

ABAP Cloud offers a variety of options to monitor the integration runtimes. With the `Application Interface Framework Message Dashboard`, the users can get an overview of the status of the SOAP and Event interfaces that they're responsible for. They can analyze the root causes of errors, and restart, or cancel messages. For the SOAP, OData, and business eventing framework, dedicated error logs are available to help the administrator during setup and operation of the integrations.

For more information, see [Administrate, Configure, and Monitor \[page 89\]](#).

Design-Time for Restriction of Data Access and Activities for Integration Services

Note

This paragraph gives only a short overview of relevant considerations for access management for transactional and services. For a holistic overview, see [Identity and Access Management](#). For more details about the conceptual background for access management, see [Conceptual Background: Access Management \[page 41\]](#).

Depending on the protocol type, the domain-specific logic of an Integration Service either uses the architecture of the RAP programming model (OData, SQL) or individual ABAP implementations (HTTP, RFC). When the RAP programming model architecture is used, the authorization checks are triggered automatically by the authorization handler of the behavior definition and access control of CDS entities. If the integration service is implemented without RAP, the authorization checks can occur in all parts of the implementation, either by explicit authority check statements or by calling APIs (for example CDS entities or behavior definitions) with their own authorization checks.

Develop Integration Services

With integration services, you develop system-to-system or event-based integration.

For more information, see:

- [Develop Integration Services \[page 63\]](#)
- [Develop Tests \[page 87\]](#)
- Configuration Time: [Administrate, Configure, and Monitor \[page 89\]](#)

3.2 Develop

This chapter explains the requirements and considerations during the develop phase for each programming model aspect.

Develop Apps, Extensions, and Services with ABAP Cloud

You can develop new apps and services from scratch, or extend existing SAP or custom services in an upgrade-safe way. To create a new app and services, you implement the domain-specific implementation, including the respective data model and business logic, according to the selected use case. All domain-specific data

models that are based on a RAP architecture blueprint will bring extensibility as a built-in quality. Generally, extensibility is based on an opt-in approach, meaning that the original data model or service must be enabled for the various extensibility use cases.

Accessing Data

Data for the domain-specific implementation can originate from an SAP HANA database or a remote integration partner. In a remote setup, the data is consumed in an outbound communication scenario. For more information about how to implement an outbound communications scenario and the available protocols, see [Consume External Integration Services \(Outbound Communication\) \[page 70\]](#).

Develop an App from Scratch

Once you've decided on the respective use case with a programming aspect, you can implement the domain-specific models and the domain-specific logic. The domain-specific data model implements the design-time with the respective development objects.

For more information, see [Develop SAP Fiori Apps \[page 51\]](#).

Develop Integration Services

There are integration services provided by SAP to integrate different products out-of-the-box. The available scenarios depend on the respective SAP product. You can also develop your own integration services, for example, to use them as a data source or to implement other custom process and data integration use cases. The details outlined in this chapter focus on developing custom integration services to integrate apps and processes for end-to-end use cases.

For more information, see [Develop Integration Services \[page 63\]](#).

Extend an Existing App or Service

Extensibility is required when you want to extend the scope of an original SAP or custom app or service. The available extensibility options always depend on the extensibility-enablement of the original app or service. It isn't possible to implement extensions beyond the enabled scope. For more information, see these topics:

- [Extend Transactional Services \[page 74\]](#)
- [Extend Analytical UI Services \[page 77\]](#)

The available extensibility options and frameworks depend on the respective SAP product. For an overview, see [ABAP Cloud and Different Extensibility Options \[page 26\]](#).

What's Next

Once you've developed your app or service, you can develop automated tests for continuous quality assurance: [Test \[page 83\]](#).

[Develop SAP Fiori Apps \[page 51\]](#)

This chapter explains how to develop Fiori apps with ABAP Cloud.

[Develop Integration Services \[page 63\]](#)

Integration services enable seamless data exchange and business process coordination.

[Develop Extensions \[page 73\]](#)

This chapter explains how to extend existing services and apps.

[Develop with Asynchronous Programming Patterns \[page 79\]](#)

Asynchronous communication enables applications to execute operations independently, promoting efficiency and performance.

[Develop Localization and Internationalization \[page 81\]](#)

This chapter explains how apps and services can be translated and adapted to country/region-specific requirements.

[Document Apps and Services \[page 82\]](#)

This chapter explains how to document apps and services.

3.2.1 Develop SAP Fiori Apps

This chapter explains how to develop Fiori apps with ABAP Cloud.

About SAP Fiori Apps

SAP Fiori apps provide a user-centric interface for user. With ABAP Cloud, you can offer a responsive design and a holistic user experience across different devices like desktops, tablets, and mobiles.

Develop Authorizations

With authorizations for *transactional services*, you determine which users have rights to perform certain actions or to access-specific data within the application.

For *analytical services*, you implement access management with access controls in CDS as part of the multidimensional data model.

For background information about access management, see [Conceptual Background: Access Management \[page 41\]](#).

For more information, see the following topic:

- [Develop Authorizations \[page 55\]](#)

Develop Transactional Services

The core of the domain-specific model in the ABAP RESTful Application Programming Model is the RAP business object. A RAP business object consists of a structure modeled with ABAP Core Data Services, a behavior defined in the behavior definition and the respective behavior implementation. From a structural point of view, a business object consists of a hierarchical tree of nodes where the nodes are linked by compositions.

The domain-specific logic that's used to define the behavior for a RAP business object is implemented with ABAP and CDS data definition language. Depending on your use case, RAP BOs brings standard operations, such as create or update out-of-the-box and offer possibilities to develop additional behaviors, such as determinations, validations, or actions.

Typical use cases for transactional apps include apps to administer master data, create, and maintain business configurations, or to accommodate any use case that requires reading and modifying data.

For more information about developing transactional apps, see these topics:

- [Develop Transactional UI Services \[page 55\]](#)
- [Develop Transactional Services for Business Configuration \[page 57\]](#)

Develop Analytical Services

The core of the domain-specific model for analytical services is the analytical provider and the analytical query. The analytical provider defines the multidimensional data model and KPIs that are used for analytical reporting. Analytical queries define the initial reporting layout, and can be used to perform various analytical evaluations and calculations.

For more information, see [Develop Analytical UI Services \[page 60\]](#).

Develop UIs

End users need a User Interface (UI) to use inA-based analytical services and OData-based transactional services. You can define the UI layout together with the data model. You can use UI annotations for transactional apps and analytical annotations for analytical apps. In addition, you can further refine the UI with SAP Business Application Studio and SAP Fiori tools.

SAP Business Application Studio is an SAP Business Technology Platform (SAP BTP) service that offers a modern development environment that's designed for efficient development of business applications for the SAP Intelligent Enterprise. For more information, see [SAP Business Application Studio](#).

SAP Fiori tools is a set of extensions for SAP Business Application Studio that makes developing SAP Fiori applications faster and easier. For more information, see [SAP Fiori Tools](#).

Make the App Available on the Launchpad

To enable business users to use an app or service, the final service and its UI need to be available on the SAP Fiori launchpad. For more information, see [Administrate, Configure, and Monitor \[page 89\]](#).

3.2.1.1 Transactional Services

With the The ABAP RESTful Application Programming Model, you can develop and extend transactional lifecycle-stable and upgrade-safe apps and services.

About Transactional Services

Transactional Services in ABAP Cloud provide a robust and scalable solution for managing business processes and data. You can define data models, implement business logics using a declarative approach, reducing development effort and increasing productivity.

The ABAP RESTful Application Programming Model (in short RAP) defines the architecture for efficient end-to-end development of intrinsically SAP HANA-optimized OData services for transactional use cases. RAP supports the development of all types of Fiori apps, including master data or configuration apps.

You can build transactional services that support complex standard and nonstandard operations, such as create, read, update, and delete (CRUD), actions or determinations and validations on business objects. These services can be easily consumed by external systems, mobile applications, or other ABAP-based applications, enabling seamless data exchange and process automation.

The RAP architecture is also the basis for remote and local business events. For details about business events, see [Develop Event-Based Integration \[page 72\]](#).

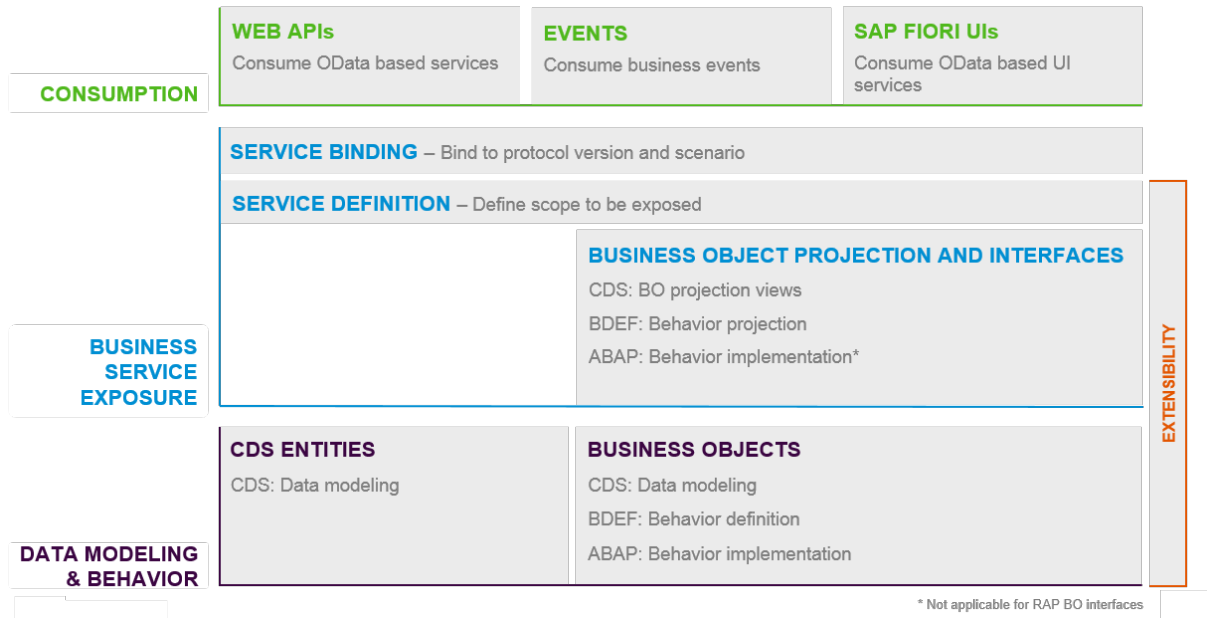
Details About the Transactional Design Time

You use ABAP Core Data Services (CDS) to define the domain-specific data model for a RAP Business object. The different nodes of a RAP BO are modeled in a compositional hierarchy. The behavior is defined in the behavior definition, and implemented with ABAP and the `Entity Manipulation Language` in the ABAP behavior pool of a RAP Business object. BO projections and interfaces allow you to define additional abstraction layers on top of the data model. Transactional services can be exposed as UI service, business event, or Web API.

For more details about the design time, see:

- [Business Object](#)

RAP - The big picture



- [Data Modeling and Behavior](#)
- [Business Object](#)
- [Query](#)
- [Business Service](#)
- [Service Definition](#)
- [Service Binding](#)
- [OData Service Consumption](#)
- [Develop Web APIs](#)
- [OData Service Consumption](#)
- [Business Object Projection](#)
- [Business Events](#)

[Develop Transactional UI Services \[page 55\]](#)

RAP offers different implementation types to adapt the development process to your business requirements.

[Develop Transactional Services for Business Configuration \[page 57\]](#)

Business configuration includes the development, maintenance, and delivery of objects and data that is the basis for business processes or standard code lists - for example, ISO standards, industry, localization, or best practices.

3.2.1.1.1 Develop Transactional UI Services

RAP offers different implementation types to adapt the development process to your business requirements.

Transactional Data Models

Transactional data models in ABAP Cloud are designed to handle data transactions in a way that ensures data consistency, integrity, and reliability. This data model is built around the concept of encapsulating actions that read or write data into distinct, atomic units of work known in the SAP LUW. This facilitates real-time processing of business transactions, while maintaining concurrency control to ensure data integrity. The model-driven approach ensures that the behavior of business objects is adaptable to business requirements, with dedicated exits for behavior implementation.

Furthermore, you can natively implement application-to-application integration, for example, with reuse services or business events, or other services such as workflows.

(Behavior) Building Blocks for Transactional UI Services

Generally, transactional services in RAP consist of a data model that's implemented with CDS, and behavior that's implemented with ABAP and the `Entity Manipulation Language` (EML). For a high-level explanation of the design-time architecture, see [Detail about the Transactional Design Time \[page 53\]](#).

This section outlines the most important building blocks that are defined in the behavior definition.

Deciding on Implementation Types

The implementation type defines the degree of standardized compared to customized behavior implementation in a RAP business object.

The main approaches are the two standard implementation types of RAP business objects: **managed** or **unmanaged**.

Both approaches are based on a data model that is defined in CDS view entities. Because the standard behavior is provided by a RAP-managed provider when using the implementation type managed, it's the developer's task to implement the complete behavior in the unmanaged implementation type. A RAP BO can only be managed or unmanaged, a simultaneous use of both implementation types isn't possible.

Note

It's recommended to use the managed implementation type whenever possible. In unmanaged use cases, you have to implement the CUD operations, and ensure that your implementation is compliant with the RAP contract. In managed use cases, this is taken care of by the RAP framework. For more information about the RAP business object contract, see [RAP Business Object Contract](#).

For more information about the implementation types, see [Business Object Implementation Types](#).

Developing Authorization Control

To define who is allowed to read or change the data of a business object, RAP offers an authorization concept to restrict access to the business object. Authorization control is always relevant when the permission to run

an operation depends on which consumer wants to run the operation. You can define authorization checks for update operations - for example, create or custom operations such as actions on **global** and **instance** level. Instance authorizations check for permissions that are related to the respective instance, and global authorization check for permissions beyond the instance scope, such as user or system-related permissions.

- **Global Authorization:** Is used for all authorization checks that depend only on the user. You can define a global authorization to check if users are allowed to run an operation.
- **Instance Authorization:** Is used for all authorization checks that, in addition to the user role, depend on the state of the entity instance in question. With instance authorization, you can define an authorization that depends on a field value of the instance.

For more information, see:

- For initial design considerations: [Transactional Use Cases - Initial Considerations \[page 43\]](#)
- [Authorization Control](#)
- [Authorization Implementation](#)

Developing Concurrency Control

Concurrency control prevents concurrent and interfering database access from different users. This ensures that data can only be changed if data consistency is assured. In RAP, you can either implement optimistic or pessimistic concurrency control:

- **Optimistic Concurrency Control:** Optimistic concurrency control enables transactional access to data by multiple users while avoiding inconsistencies and unintentional changes of already modified data.
- **Pessimistic Concurrency Control:** Pessimistic concurrency control prevents simultaneous modification access to data on the database by more than one user.

For more information, see:

- [Optimistic Concurrency Control](#)
- [Pessimistic Concurrency Control \(Locking\)](#)

Developing Draft

📘 Note

Using draft is recommended for UI use cases to improve the user experience.

Apps with draft capabilities let users save their changes in the system and resume their work later, even on a different device, or after an unplanned shutdown. You can easily add draft to a business object with the available quick fixes that generate the required draft tables and draft actions.

For more information, see [Draft](#).

Developing Operations, Validations and Determinations

A RAP BO has standard functionality that works out-of-the-box with a managed implementation type, for example create, update, or delete operations. You can implement 'additional **operations** such as actions to develop custom modify operations - for example, to set a specific field value.

You can implement **validations** to check your data for consistency and against certain conditions based on business or process requirements. A validation is implicitly invoked by the business object's framework if a trigger condition of the validation is met. Trigger conditions can be modify operations and modified fields.

You can implement **determinations** to trigger modify operations based on trigger conditions, for example when another field is changed. A determination is implicitly invoked by the business object's framework if the trigger condition of the determination is fulfilled. Trigger conditions can be modify operations and modified fields.

For more information, see:

- [Operations](#)
- [Validations](#)
- [Determinations](#)

3.2.1.1.2 Develop Transactional Services for Business Configuration

Business configuration includes the development, maintenance, and delivery of objects and data that is the basis for business processes or standard code lists - for example, ISO standards, industry, localization, or best practices.

About Business Configuration

An important part of a business application is its business configuration. Business configuration in enterprise software refers to a predefined set of configuration options that affect the functionality and behavior of an app. Business Configuration maintenance requires a high number of apps standardized UI.

For background information about business configuration, see [Conceptual Background: Business Configuration \[page 40\]](#).

Develop Authorization Control for Business Configuration Apps

Business configuration apps follow the RAP authorization control concept to protect your business configuration object against unauthorized access to customizing data. Business configuration apps follow a generic authorization approach that can be applied to app business configuration apps.

For more information, see [Authorization Control](#).

Develop Business Configuration Apps

You have two options available to maintain business configuration content:

- [RAP-based Business Configuration Apps](#): Based on the RAP architecture blueprint, you can develop business configurations apps with a standardized Fiori UI. This is recommended for more complicated use cases that require advanced transactional capabilities. You can use the `Generate ABAP Repository`

Objects Wizard to generate the required development objects, based on a maintenance table. You must expose the RAP business object as an OData V4 – UI service and enable a draft.

- [Uploading Business Configurations](#): If no maintenance UI is required for the business configuration, you've an upload option available to maintain the content.

For more information, see:

- [Generating a Business Configuration Maintenance Object with the Generate ABAP Repository Objects Wizard](#)
- [Upload Business Configurations](#)

Check for Changes

With the Business Configuration Change Logs app, you can track content changes in your business configuration tables. You can use data change logs to check when data was changed, which kind of data was changed, or by whom.

For more information, see [Business Configuration Change Logs](#).

3.2.1.2 Analytical Services

As you know, analytics is the process of gathering, interpreting, and deriving insights from data to make informed decisions and drive improvements.

CDS defines the architecture and building blocks for ABAP analytics. CDS lets you efficiently build multidimensional data models, either in an embedded scenario, or in a side-by-side scenario for UI consumption. With CDS, you can create dimensions, facts, cubes, and hierarchies to analyze your business data, and to define measures, aggregations, and much more. For an overview of the analytical design-time, see [Analytical Use Cases - Initial Considerations \[page 45\]](#).

The following chapters focus on the embedded analytics use case and highlight the advantages of this approach.

About Analytical Services

Embedded analytics allows you to build sophisticated and complex analytical data models to evaluate and analyze business data in your ABAP system. In embedded analytics, the ABAP analytical engine is part of the software stack and operates on the same data persistence as the transactional applications. The analytical queries operate directly on the business data, without data replication to an external data warehouse system. Instead, the real-time business data is queried to always evaluate the most recent changes and trends in your business data.

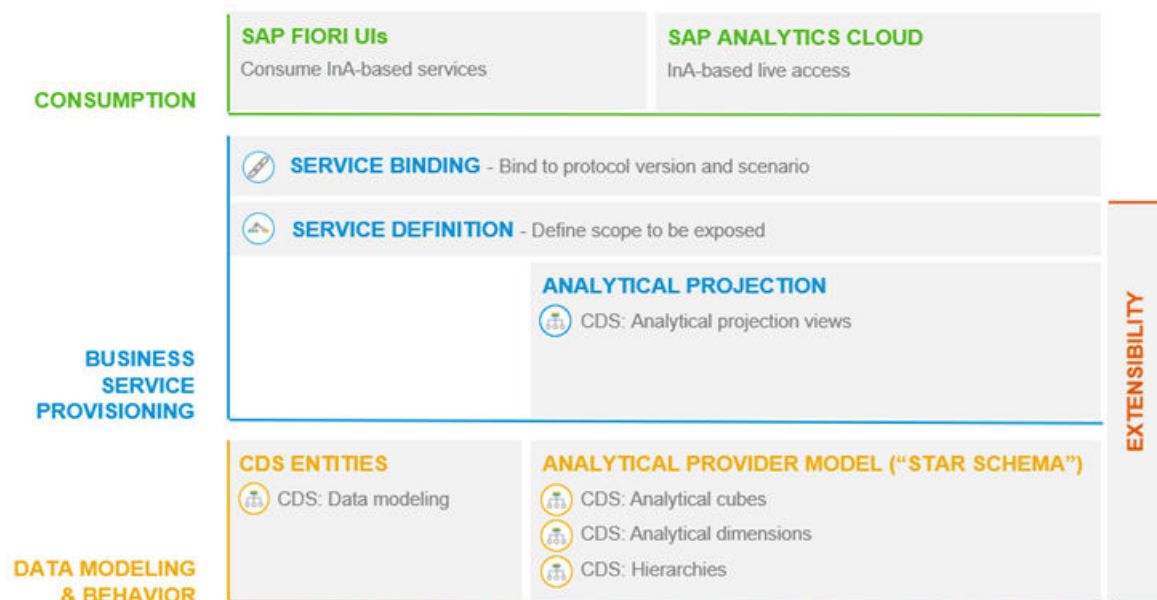
Embedded analytics unite the advantages of CDS data modeling with the analytical capabilities of the analytical engine to create real-time evaluations that are based on business data within the same ABAP system. The CDS framework is at the core of the ABAP Cloud development model, enabling the development of cloud-ready and lifecycle-stable analytical data models.

Details About the Analytical Design Time

Analytical data models rely on the same architecture blueprint as transactional RAP apps and services, easily enabling reuse and integration between both. You use ABAP Core Data Services (CDS) to model the star or snowflake schema for the analytical reporting. CDS is the basis for cubes, dimensions, and hierarchies. The aggregations are defined in the analytical query.

For more information, see:

- [Analytical Data Modeling](#)
- [Analytical Queries](#)



1

-
- [Develop Analytical UI Services \[page 60\]](#)
- [Develop Analytical UI Services \[page 60\]](#)
- [Extend Analytical UI Services \[page 77\]](#)
- [Develop Analytical UI Services \[page 60\]](#)
- [Develop Analytical UI Services \[page 60\]](#)
- [Develop Analytical UI Services \[page 60\]](#)

[Develop Analytical UI Services \[page 60\]](#)

With the Core Data Services (CDS), you can create user-friendly apps and services that enable organizations to analyze data, generate reports, and make informed decisions based on multidimensional data models.

3.2.1.2.1 Develop Analytical UI Services

With the Core Data Services (CDS), you can create user-friendly apps and services that enable organizations to analyze data, generate reports, and make informed decisions based on multidimensional data models.

Analytical Data Models

The quality of transactional data models is generally measured by their level of normalization, usually based on an entity-relationship model (ER). The ER-model offers little possibility for semantic meaning, since it's optimized for processual speed and the minimization of data redundancy. Analytical models in contrast have a different focus. Since data isn't being altered, one of the main focus points for multidimensional models is a minimal join depth and a semantically rich data model that is as easy to understand as possible.

In ABAP Cloud, the analytical programming aspect is implemented using CDS and ABAP. This allows you to build the analytical data model on the same stack and with the same technology and architecture blueprint as the transactional aspect. This allows for convergence and maximal reuse and compatibility between the analytical and transactional aspect in ABAP Cloud.

Building Blocks for Analytical Apps and Services

Developing Building Blocks of a Multidimensional Data Model

Generally, a multidimensional data model consists of the following components:

- **Dimensions:** Dimensions contain master data-like information about the individual records in a data set. They provide the contextual information of a dimensional data model, essentially describing the business context for individual measures contained in cube or a fact. An example for a typical dimension is the customer information in the case of a sales order, containing information like a customer's shipping address or other information. For more information, see [Dimensions](#).
- **Hierarchies:** Hierarchies define the result set granularity, allowing hierarchical navigation through the data set on different aggregation levels. An example is the profit earned by a city, region, or country. Each level represents a different aggregation level for the measure profit.
- **Facts:** A fact typically represents a business transaction or a business event that the analysis is based on. From a technical perspective, a fact contains the measures and foreign key relationships with a cardinality [0..1] to all relevant dimensions. In ABAP, a fact is an optional part of an analytical data model. For more information, see [Facts](#).
- **Cubes:** A data cube represents the multidimensional data model, consisting of different dimensions and measures arithmetically derived from previously defined facts. In ABAP, an analytical data model can be based directly on a cube. For more information, see [Cubes](#).
- **Analytical Query:** Analytical queries define the initial layout that is displayed, select the initial data, and can calculate measures that weren't included in the underlying cubes or facts. An analytical query must always be based on a cube. For more information, see [Analytical Queries](#).
- **Analytical Client:** The analytical client consumes the analytical data model and displays it graphically for business users. For an overview of analytical clients- see [Analytical Clients](#).

All parts of the multi-dimensional data model are based on ABAP Core Data Services.

Add Domain-Specific Information to the Multidimensional Data Model

The domain-specific information for multidimensional data models, which the analytical engine must process, is added with analytical annotations directly in the data model itself. Analytical services rely on annotations to add semantically relevant information for the consumption of multidimensional data models.

The `Analytics` annotations specify the facts (center of the star schema), extraction capabilities for replicating data into further systems, and properties of the analytical query. The `Analytics.DataCategory` annotation is used, for example, to define the CDS building blocks of the star schema. The annotations `Analytics.query: true` defines the analytical query.

`AnalyticsDetails` annotations specify the default multidimensional layout of the analytical query, the sequence of variables in UI consumption, and the specific aggregation and planning behavior of the data.

For more information, see:

- [Analytics Annotations](#)
- [AnalyticsDetails Annotations](#)

Calculate Measures and KPIs

Numeric fields, such as measures and KPIs can be aggregated based on different rule sets, such as sum or average. In ABAP Cloud, summation (SUM), minimum (MIN), and maximum (MAX) are part of the standard aggregation behavior.

The standard aggregations can be further refined on the query level with calculated measures. With calculated measures, you can define additional calculations and queries to derive additional KPIs from your measures.

With restricted measures, you can exclude certain values from the aggregation. This enables you to filter the measure result set, based on defined conditions.

For more information, see:

- [Standard Aggregations](#)
- [Calculated Measures](#)
- [Restricted Measures](#)

3.2.1.3 UIs for Analytical and Transactional Apps

UIs create an intuitive and user-friendly interface for users to interact with the business processes. In ABAP Cloud, SAP Fiori ensures a consistent user experience when developing UIs.

ABAP Cloud seamlessly integrates with Fiori and Fiori elements. Fiori elements are a framework that includes commonly used floor-plan templates, making development faster by reducing the amount of front-end code required to build SAP Fiori apps. This complements the metadata-driven UI development approach in ABAP Cloud.

About Metadata-Driven UI Development

Development in ABAP Cloud is focused on the domain-specific implementation and is thus server-driven. To integrate the UI metadata closely with the data model, the UI-specific metadata is added directly to the data model using `UI` annotations for transactional apps or analytical annotations for analytical apps.

Fiori elements consume the metadata accordingly, and create standardized UIs out-of-the-box, based on the newest floor plans and designs for analytical and transactional apps. This enables you to implement full-stack UI services without having to implement any additional front-end code, unless required by your use case. If needed, you can implement custom UI additions or extensions using `SAP Business Application Studio` and `SAP Fiori Tools`.

Developing UIs for Transactional Apps


Annotation-based UI development involves using domain-specific annotations to define the layout, data binding, and behavior of a UI. Using UI annotations allows you to define complex UIs without writing extensive code. The rendering of the UI is then handled by a runtime framework, which interprets the annotations to create the final user interface.

Transactional services use **UI annotations** to define the layout and data binding of the respective service. You can define the overall layout for standard floor plans, define labels, implement value helps or define navigation on the UI, and so on.

In addition, you can also implement back-end features that are closely related to the UI, such as feature control and side effects:

- **Feature Control:** You can provide information to the service on how data has to be displayed for consumption in an SAP Fiori UI, for example if fields are mandatory or read-only.
- **Side Effects:** You can make a Fiori Elements UI aware that data changes of defined fields require the recalculation of other data values, permissions, or messages on the UI, in the case where UI scenarios are based on draft-enabled BOs.

For more information, see:

- [SAP Fiori Element Feature Showcase App for the ABAP RESTful Application Programming Model](#) 
- [Feature Control](#)
- [Side Effects](#)

Developing UIs for Analytical Apps

Analytical services use the `AnalyticsDetails` Annotations to specify the query layout and the specific aggregation and planning behavior of the data. With the annotation `@AnalyticsDetails.query.axis: '<VALUE>'`, you can position the elements of a query on an axis and define the basic layout for the report.

For more information, see [AnalyticsDetails Annotations](#).

UI Reuse Components

For some reuse services, Fiori Reuse Components are provided and can be embedded in Fiori Elements applications. These UI reuse components complement the backend integration in your domain-specific implementation, with a ready-to-run frontend artefact. Two examples are the change document or application log reuse components, which can be embedded in your frontend with little effort.

For more information about reuse services, see [Reuse Services and Libraries \[page 29\]](#).

3.2.2 Develop Integration Services

Integration services enable seamless data exchange and business process coordination.

About Integration Services

Integration services allow you to create connections to other systems using different protocols. This enables seamless communication and data exchange between services and apps, and allows you to build end-to-end business processes across system boundaries. ABAP Cloud supports integration with various integration partners: SAP cloud products, SAP BTP services, customer extensions, external services, and the on-premise landscape of the customer through the SAP Cloud Connector.

For background information about communication management with integration services, see [Develop Integration Services \[page 63\]](#).

Decide on an Integration Use Case

Depending on whether you want to implement data integration or process integration, you have different options available.

Process Integration

Process integration can structure communication at the app level, along with a predefined business process, for example, order-to-cash. The process integration takes the domain-specific implementation of each app into account. Communication between the communication partners for point-to-point integration can be bidirectional, meaning that information is exchanged in both directions. There are both synchronous and asynchronous process integration patterns, for example event-based integration follows an asynchronous approach.

Data Integration

Data integration addresses the data exchange between two or more communication partners without a relation to a business process. One of the main use cases for data integration is cross-system analytics.

In data integration scenarios, the raw data is exchanged without leveraging the domain-specific logic.

Data can either be replicated or federated in a data integration scenario:

- **Data Federation:** Data remains in its source system and all requests are forwarded and run on the original data. Data Federation is used in scenarios where live access to the source system is required, for example to access the most recent data without a time lag, or to rely on data access control mechanisms inside the source system.
- **Data Replication:** Data is replicated to a second system and requests are run locally on the replicated data such as, for example, in a data warehouse. Data replication is used where it is more efficient to transfer (business) data from a source to a target system before the data is accessed within that target system.

📘 Expand the following table for more information.

Comparing Data Federation and Data Replication

Characteristics	Data Federation	Data Replication
Scenario Scope	<ul style="list-style-type: none"> • Data remains in its source system. • Requests are delegated to the respective source systems and run there. 	<ul style="list-style-type: none"> • Data is replicated to another system. • Requests are run locally in the respective systems. • Data transformations and adaption is often applied (data enrichment, cleaning, consolidation, multiplexing, scripting, and so on)
Consistency	<ul style="list-style-type: none"> • Transactions spanning multiple statements (Stateful transactions) 	<ul style="list-style-type: none"> • Stateless transactions
Access Control	<ul style="list-style-type: none"> • Authorizations are checked locally in each system. • Federation can be implemented with privileged mode in technical scenarios for example for the initial load. 	<ul style="list-style-type: none"> • Data replication is done in privileged mode. Authorizations are either replicated or newly defined on consumer-side.
Protocol	<ul style="list-style-type: none"> • SQL 	<ul style="list-style-type: none"> • SQL + Replication Middleware

Data and Process Integration Comparison

The following table gives you an overview of both integration service scenarios, outlining the main differences and main use cases.

📘 Expand the following table for more information.

Comparing Process and Data Integration

Dimensions	Process Integration	Data Integration
Scenario Scope	<ul style="list-style-type: none"> • Integrates several apps, along a predefined business process • Domain-specific logic is taken into account for all operations. • (Potentially) Bidirectional communication between communication partners 	<ul style="list-style-type: none"> • Integrates several communication partners without relation to any specific business process. • Domain-specific logic is bypassed (only READ logic is considered). • One-directional communication between communication partners

Dimensions	Process Integration	Data Integration
Domain-Specific Implementation	<ul style="list-style-type: none"> Write/Updates always through domain-specific implementation layer (logic is considered). 	<ul style="list-style-type: none"> Write/Updates always directly on database layer.
Data Representation	<ul style="list-style-type: none"> Exchanged data is converted to an external format on provider side (for example ISO Currencies). 	<ul style="list-style-type: none"> Exchanged data keeps source types (for example ABAP dates) to avoid any loss of data or semantics because of conversions. Only minimal conversions are applied, which can only be processed on provider-side (for example decimal shift).
Protocol	<ul style="list-style-type: none"> OData SOAP Business Events Remote Function Call 	<ul style="list-style-type: none"> SQL

Choose an Integration Protocol and Communication Pattern

Available Integration Protocols



The goal of process integration is to facilitate communication between multiple apps within a predetermined business process. ABAP Cloud offers many different protocols for integration, for these reasons:

- The communication partner (SAP products or 3rd party) usually exposes a functionality only through a dedicated protocol.
- Different use cases for integrations require different protocols, for example for asynchronous integrations or for using event-driven architectures.

The following table gives you an overview of the available protocols:

Protocols for Point-To-Point Integration

Protocol Name	Description	When To Use	Features	Constraints	Available for Inbound Communication	Available for Outbound Communication
OData	Open Data Protocol (OData) is an open protocol that allows the creation and consumption of queryable and interoperable REST APIs in a simple and standard way.	<ul style="list-style-type: none"> • Synchronous process integration with modern ABAP systems 	<ul style="list-style-type: none"> • Standard-based protocol and data format • Batch requests • System query options 	<ul style="list-style-type: none"> • Limited asynchronous options 	Yes	Yes
SOAP (Simple Object Access Protocol):	SOAP services are Web services that enable data exchange based on a standardized XML format through the Simple Object Access Protocol.	<ul style="list-style-type: none"> • Process integration with existing APIs 	<ul style="list-style-type: none"> • Standard-based protocol and data format • Synchronous and asynchronous options • Support for exactly-once-in-order pattern • AIF interfaces for monitoring of asynchronous processing 	<ul style="list-style-type: none"> • Supports only the generation of service consumers from external metadata. • No native integration with RAP 	No	Yes

Protocol Name	Description	When To Use	Features	Constraints	Available for Inbound Communication	Available for Outbound Communication
Remote Function Call (RFC)	RFC is a proprietary SAP interface for communication between ABAP systems RFC Connectors  .	<ul style="list-style-type: none"> • Synchronous process integration with old ABAP systems or for high-performance requirements • Data replication with old ABAP systems 	<ul style="list-style-type: none"> • Client libraries available • Compatibility with old ABAP releases • High performance 	<ul style="list-style-type: none"> • No support for exactly-once-in-order patterns (bgRFC) in ABAP Cloud • No integration with RAP • Only integration with ABAP or RFC Connectors . 	Yes	Yes
SQL	The ABAP SQL Service can be used to expose CDS view entities to external, ODBC-based SQL clients.	<ul style="list-style-type: none"> • Synchronous data integration interface for data federation and replication 	<ul style="list-style-type: none"> • Client libraries available • Consistency and performance for large payloads 	<ul style="list-style-type: none"> • No standardized data serialization format • Only consumption through ODBC Driver 	Yes	No
HTTP	You can use the full flexibility of HTTP protocol by using the HTTP client library to send a request to any HTTP endpoint, or provide your own HTTP service for other integration partners.	<ul style="list-style-type: none"> • Full flexibility of HTTP protocol 	<ul style="list-style-type: none"> • Flexibility to adapt implementation to specific use case requirements 	<ul style="list-style-type: none"> • No integration with RAP • No standardized data serialization format 	Yes	Yes

Name	Description	When To Use	Features	Constraints	Available for Event Provider	Available for Event Consumer
Business Events	Event-driven architecture enables asynchronous communication between an event provider and consumer in use cases where no direct response from the consumer is required.	<ul style="list-style-type: none"> Asynchronous process integration, using event-driven architectures 	<ul style="list-style-type: none"> Asynchronous integration Standard-based data format Customized payload, with derived events based on SAP delivered events AIF interfaces for monitoring of asynchronous processing 	<ul style="list-style-type: none"> Only for integration through event-driven architectures No support for in-order processing 	Yes	Yes

Expose Integration Services

When exposing integration services in ABAP Cloud, you make them available for consumption in other systems through an inbound Communication Scenario.

For more information, see [Expose Integration Services \(Inbound Communication\)](#) [page 69].

Consume External Integration Services

Consuming integration services in ABAP Cloud involves interfacing with various protocols to enable data communication. To simplify the implementation of a remote call, you can create a service consumption model for the external service for OData, SOAP, and RFC.

For more information, see [Consume External Integration Services \(Outbound Communication\)](#) [page 70].

Develop Event-Based Integration

Event-based integration is designed to listen for certain events, such as when a new item is created, and then trigger specific actions or processes in response, ensuring the process flow across systems.

For event-based use cases, you can use local events for triggering follow-up actions within the same system. With remote business events, you can design cross-system process flows.

For more information, see [Develop Event-Based Integration \[page 72\]](#).

Create Communication Arrangements (Administrator)

As a developer, you can create and expose services for external outbound consumption to a communication partner by creating an outbound service of the required type. To make the service available for consumption, you need to create a Communication Scenario and assign these services to it.

An administrator then creates a communication system and a user account for the communication partner, then maintains a communication arrangement for the scenario using the created communication system, and specifies the authentication method and URLs.

3.2.2.1 Expose Integration Services (Inbound Communication)

This chapter outlines how to develop integration services for inbound communication.

ABAP Cloud lets you exposure of business services to other systems through inbound communication. To make an integration service externally available, you must do the following:

- **Develop an Integration Service for Process or Data Integration:** Create an integration service to process the incoming request and to create the response for the communication partner.
- **Set up Communication Management:** Prepare administrative setup for the administrator by creating, for example, a communication scenario.

Develop Integration Services for Process Integration

ABAP Cloud supports various protocols such as HTTP, remote function calls, or services published through service bindings, such as OData for process integration.

Develop OData Web APIs

An OData API is an OData service whose metadata doesn't entail any UI-specific annotations that are defined for the data model. An OData API facilitates the exchange of business information between an application and a client. OData APIs are defined with the design-time architecture of the ABAP RESTful Application Programming Model.

For more information, see [Develop Web APIs](#).

Develop HTTP Services

You can develop an HTTP service by creating an HTTP service object. The required handler class to handle the http request is automatically created with the HTTP service object. The interface

`IF_HTTP_SERVICE_EXTENSION`, with HTTP request/response parameters, enables you to build an HTTP service with full flexibility.

For more information, see:

- [HTTP Service Development](#)
- [Working with the HTTP Service Editor \(ABAP Development Tools: User Guide\)](#)

Develop RFC Services

For more information, see [RFC - Inbound Communication](#).

Develop SQL Services for Data Integration

You can access CDS view entities in an ABAP system using SQL and the open database connectivity (ODBC), a standard API for accessing databases. As a result, you can use SQL statements in external analytical tools to access data in database tables that reside in an ABAP system. You can create an RFC service based on a remote-enabled functions module. You have full flexibility with the implementation details.

For an example, see:

- [Developing and Exposing an SQL Service in the ABAP System](#)
- [Exposing the SQL Service for Privileged Access](#)
- [Exposing the SQL Service for Business User Access](#)

Setting up the Communication Management for Inbound Communication

You can create an RFC service based on a remote-enabled functions module. You have full flexibility with the implementation. You have to create a communication scenario with inbound services for the exposed endpoints of the communication partner. This enables the administrator of the system to set up the integration. For example, the administrator can create credentials for the authentication of the communication partner and assign corresponding authorizations to the communication user in the `Communication Management` apps.

For more information, see [Consuming Services in the Context of API with Communication Users \(Inbound\)](#).

3.2.2.2 Consume External Integration Services (Outbound Communication)

This chapter outlines how to set up outbound communication to enable the consumption of services from other systems.

Not all data that is consumed in ABAP Cloud is read from the SAP HANA database. Instead, data can be consumed using outbound communication through many protocols from remote integration partners, both from cloud services and from the on-premise landscape. For outbound communication, you do the following:

- **Implement the Business Service Consumption:** Create a proxy class to create the request and processing the response by the communication partner.
- **Set up the Communication Management:** Prepare integration setup for the administrator by creating, for example, a communication scenario.

Consume Services for Process Integration

To simplify the implementation of a remote call, you can create a service consumption model for the external service. The service consumption model creates typed proxies for the remote service. That way, you can access the service in a strictly typed process without the need to compile requests and parse responses.

Consume OData Services

The OData client proxy is the interface between the client (consumer of a service) and the service implementation (data provisioning) in the OData service consumption in ABAP. The OData client proxy enables you to create an OData typed proxy to run OData requests in your ABAP implementation.

For more information, see [OData Services](#).

Consume HTTP Services

You can implement free-style integrations without generated proxies via the HTTP client library.

For more information, see [Enable HTTP Communication in Your ABAP Code](#).

Consume Remote Function Call Services

You can generate an ABAP proxy for calling one or more remote-enabled function modules (RFMs) using a service consumption model with a typed proxy. You can also directly use the `CALL FUNCTION . . . DESTINATION` statement in your implementation.

For more information, see:

- [RFC](#)
- [Generating Proxies for Remote Function Call \(ABAP Development Tools: User Guide\)](#)

Consume SOAP Services

You can generate a service consumption model that is based on a Web service description language (WSDL) file that describes your service . With this service consumption model, you can consume SOAP services using the typed proxy.

For more information, see [SOAP](#).

Consume SQL Services for Data Integration

After installing the ODBC driver for ABAP, you can use SQL service and the ODBC driver to provide data access from external ODBC-based clients.

For more information, see:

- [Consumption of the SQL Service and the ODBC Driver for ABAP](#)

Set Up Communication Management for Outbound Services

Create a communication scenario with outbound services for the endpoints of the communication partner. This enables the administrator of the system to set up the integration by maintaining the credentials for authentication at the communication partner in the Communication Management apps. For more information, see [Consuming Services in the Context of API with Communication Users \(Outbound\)](#).

ABAP Cloud supports complex scenarios, where the relevant integration partner is determined at runtime (receiver determination). In these cases, the administrator can set up the communication scenario multiple times in each tenant.

3.2.2.3 Develop Event-Based Integration

ABAP Cloud supports event-driven architecture natively, without point-to-point integration.

About Business Events

Event-driven architecture enables asynchronous communication between an event provider and an event consumer in use cases where no direct response from the event consumer is required. Events represent a significant change of state that is relevant for follow-up processes such as, for example if a new travel is created and you want to enable consuming applications to trigger additional workflows.

This chapter focuses on remote business events. Local business events are based on the same design-time, but are consumed using an event handler class that is set up as a class pool, similar to a behavior pool. For more information about consuming local events, see [Local Consumption](#).

Expose Business Events

The development of business events is natively integrated with the ABAP RESTful Application Programming Model. An event, that is defined in a RAP behavior definition, can be exposed to remote consumers by assigning an event binding. The event binding defines the topic of the event.

The administrator can connect the system to an SAP Event Mesh instance and expose this event topic in the `Maintain Event Channel Binding` app. Potential integration partners can then consume this event topic from the Event Mesh.

For more information, see:

- [Business Events](#)
- [Develop Business Events](#)

- [Checking Channel Binding](#)

Consume Business Events

To consume events from other systems remotely, you can generate an Event Consumptions Model based on the AsyncAPI metadata of the event. The Event Consumption Model generates a typed handler class to process the event.

To simplify the administrator's tasks, multiple Event Consumption Models can be bundled into a communication scenario. The administrator can then create a communication arrangement to link a channel to an SAP Event Mesh with a communication scenario, so that the list of event types is processed by a communication user.

For more information, see:

- [Business Event Consumption](#)
- [Generating an Event Consumption Model](#)

3.2.3 Develop Extensions

This chapter explains how to extend existing services and apps.

About Extensions

The extensibility concepts in ABAP Cloud are built on the release contract framework and a cloud-optimized ABAP language. This combination guarantees that extensions can be made to ABAP Cloud apps and services, without needing to modify the existing code. This approach ensures that extensions are stable and provides multiple options for extending apps and services.

The chapters focus on extensibility in the scope of developer extensibility. For an overview of different extensibility options and their availability in different products, see [Extensibility \[page 26\]](#).

For background information about the release contract framework and the ABAP language, see:

- [Public Released APIs \[page 24\]](#)
- [Cloud-Optimized ABAP Language \[page 23\]](#)

Checking the Data Model or Behavior for Possible Extensibility Options

In ABAP, extensibility is implemented using an opt-in approach. This means that when implementing the data model or behavior, specific extensibility options need to be added. By default, data models and behavior in ABAP Cloud aren't extensible. Therefore, if you want to extend a service that's provided by SAP or a custom

service, you need to first determine which extensibility options are available for the respective app or service. Then can you start developing the extensions.

For more information, see [Extend SAP S/4HANA in the cloud and on premise with ABAP based extensions](#) .

Extend Services and Apps

Once you've checked the available extensibility options, you can extend a transactional service with, for example, additional fields or new behavior such as new actions, functions, or validations. You can extend analytical services with new dimension and hierarchies.

For more information, see:

- [Extend Transactional Services \[page 74\]](#)
- [Extend Analytical UI Services \[page 77\]](#)

3.2.3.1 Extend Transactional Services

You can easily extend your transactional data models in an upgrade-safe and lifecycle-stable way.

Extensibility is a built-in quality for all apps and services that are developed with ABAP Cloud. For more general information about extensibility, see [Extensibility \[page 26\]](#). This paragraph outlines specifically how the built-in quality is implemented for apps and services developed with RAP.

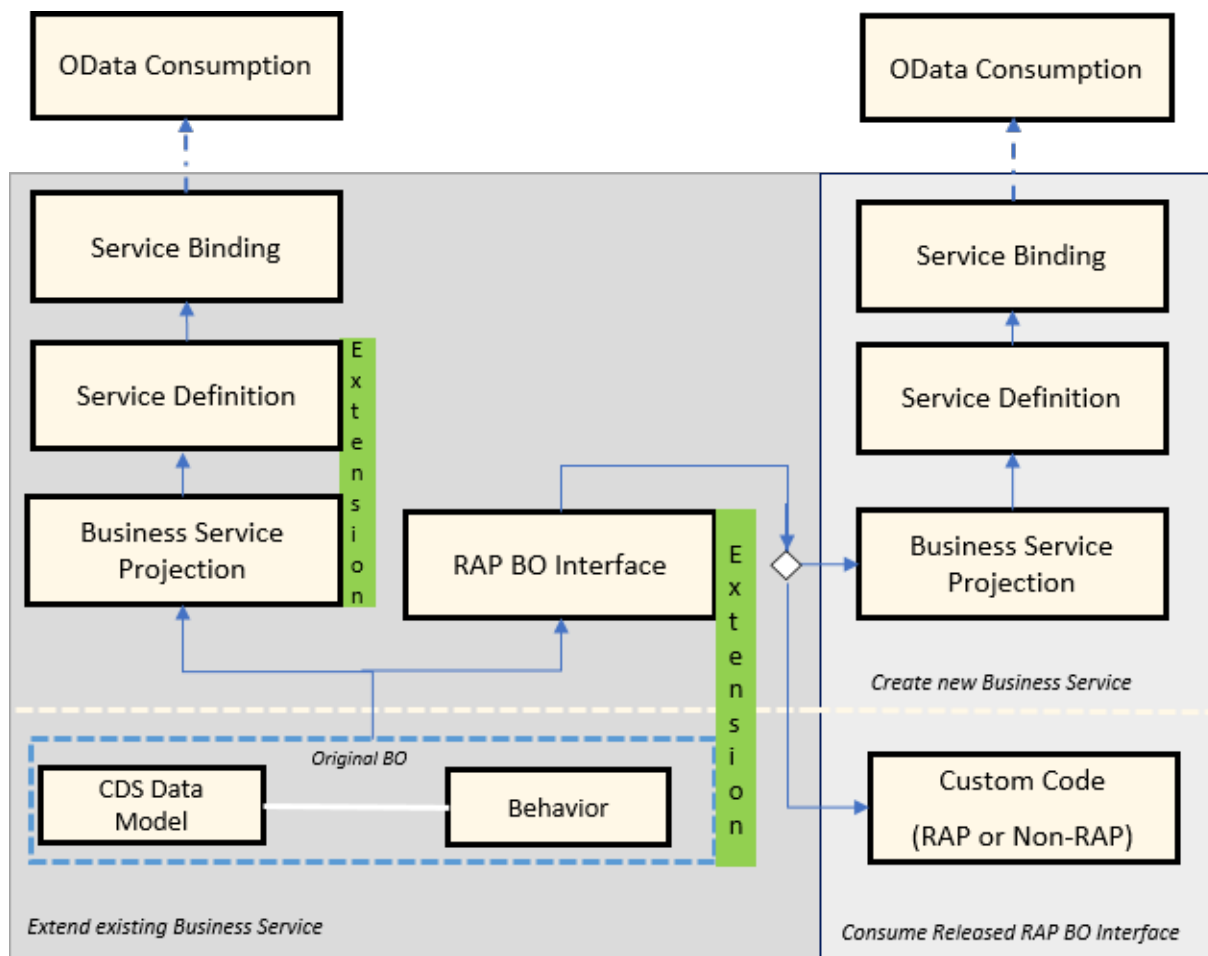
For an overview of personas involved in RAP extensibility, see [Relevant Personas in ABAP Cloud \[page 10\]](#) and [Extend](#).

About RAP Extensibility

RAP Extensibility offers the possibility to develop semantically rich, upgrade-safe, and lifecycle-stable extensions for RAP business objects on each layer of the RAP stack. An original business object developed by a BO provider is extended with additional functionality to extend the functional scope of the original RAP BO. Using well-defined extension points enabled in the original business object, an extension provider can then extend the original data model and behavior in accordance with the business requirements or create their own business service based on a RAP BO interface.

For an overview of the extensibility architecture, see [Extensibility Architecture Overview](#).

RAP Extensibility Big Picture



Generally, the RAP extensibility architecture is based on an *Opt-in* approach to provide fine-granular control over which parts of a business object can be extended. This means, that every possible extension point must be defined explicitly in the corresponding development objects in the data model and the behavior definition and implementation. Once enabled, every BO layer can be extended with one or several extensions from extension provider side. In addition, the extensibility architecture ensures a separation of concerns between original BO elements and extensions to limit technical dependencies to public and well-defined extension points. This guarantees lifecycle-stability and upgrade-safety that is also checked and technically enforced by the RAP framework.

Extension providers can extend RAP BOs and their services on-stack or develop and expose their own services based on released RAP BO interfaces.

Extensibility Use Cases

Extensibility Use Cases

Extensibility Use Case	Extensibility Persona	Background Information
Data Model Extension: Build full-stack data model extensions by adding new fields and associations including corresponding behavior characteristics and authorization control.	Extensibility-Enabler: Adds annotations and extension include structures to the original RAP BO to enable data model extensions.	<ul style="list-style-type: none"> For more information about enabling full-stack data extensibility, see Extensibility-Enablement for CDS Data Model Extensions. For an implementation example, see Enabling Data Model Extensions.
	Extension Provider: Extends the original RAP BO with new fields or associations including field characteristics depending on the options defined by the extensibility-enabler.	<ul style="list-style-type: none"> For more information about how to develop data model extensions, see CDS Data Model Extensions. For an implementation example, see Develop Data Model Extensions.
Behavior and Field-Related Behavior Extensions: Build additional behavior like new validations, determinations, or actions including dynamic feature control and other field-related behavior.	Extensibility-Enabler: Enables data model extensibility and behavior extensibility on the original RAP BO.	<ul style="list-style-type: none"> For more information about enabling your BO for behavior extensions, see Extensibility-Enablement for Behavior Extensions. For an implementation example, see Enabling Non-Standard Behavior and Field-Related Behavior and Enabling Standard Behavior Extensions.
	Extension Provider: Extends the original RAP BO with new validations, determinations, or actions depending on the options defined by the extensibility-enabler.	<ul style="list-style-type: none"> For more information about how to develop different behavior extensions, see Behavior Extensions. For an implementation example, see Develop Behavior Extensions.
Node Extensibility: Build additional BO nodes with own behavior and data model with node extensibility.	Extensibility-Enabler: Enables node extensibility on the original RAP BO.	<ul style="list-style-type: none"> For more information about node extensibility enabling, see Extensibility-Enablement for Node Extensibility. For an implementation example, see Enabling Node Extensions.
	Extension Provider: Extend the original BO with new nodes that have their own data model and behavior.	<ul style="list-style-type: none"> For more information about how to develop node extension, see Node Extensions.

Availability of Extensibility as Part of Different Products

All extensibility options are built on stable public extension points, which remain unaffected and consistent through upgrades and changes. For more information about how ABAP Cloud relates to different extensibility options, see [How ABAP Cloud Related to Different Extensibility Options \[page 6\]](#).

For an overview of the different extensibility options, see [ABAP Cloud and Different Extensibility Options \[page 26\]](#).

3.2.3.2 Extend Analytical UI Services

CDS allows you to extend your analytical data models while ensuring their compatibility with upgrades and maintaining their stability throughout their lifecycle.

Extensibility is a built-in quality for all apps and services developed with ABAP Cloud. For more general information about extensibility, see [Extensibility \[page 26\]](#). This paragraph outlines specifically how the built-in quality is implemented for analytical apps and services developed with CDS.

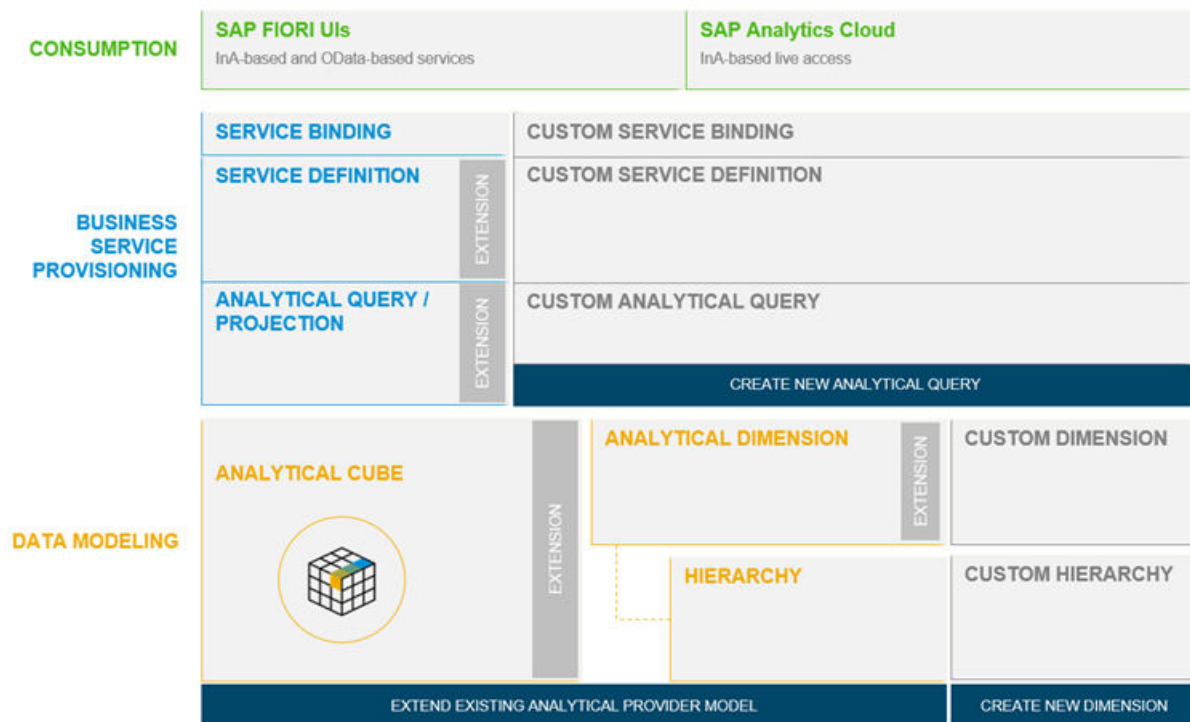
About Extensibility for Analytical Providers

Extensibility is a built-in quality for all ABAP data models based on CDS. Based on an opt-in approach, the entire data model stack can be enabled for different extensibility use cases, depending on the specific data model requirements. Each layer in the stack offers possibilities for dedicated extension points, enabling a well-defined separation of concerns between the original data model and the extensions.

Extensibility for Analytical Providers Big Picture

In an analytical data model, each part of the data model can be extended individually using predefined extension points in the original analytical data model. You can use extension points to make your own analytical data models extensible, or to extend predelivered SAP content. All parts of the analytical provider can be extended. You can, for example, add new dimensions to a cube to enhance the reporting depth, or you can add additional hierarchies to enable additional navigation throughout the data set.

The following graphics show you where analytical data models can be enabled for extensibility.



Extensibility Use Cases

Extensibility Use Cases

Extensibility Use Case	Extensibility Tasks
Dimension Extension <p>Add new hierarchies to a dimension, or add new fields or associations to the dimension to diversify the data model.</p>	Extensibility Enablement <p>Add annotations and extension, include structures to the original dimension to enable data model extensions such as fields or associations.</p> <hr/> Extension Development <ul style="list-style-type: none"> Add new hierarchy: To extend a dimension with a new hierarchy, create a new hierarchy and add it as an association to the corresponding dimension. Add new field extension: Extend the original dimension with additional fields.
Cube Extension <p>Add new dimensions to a cube to extend the scope of the data analysis or add new measures to the cube to calculate additional values.</p>	Extensibility Enablement <p>Add annotations and extension, include structures to the original data model to enable data model extensions, such as fields or additional dimensions.</p>

Extensibility Use Case	Extensibility Tasks
	Extension Development <ul style="list-style-type: none"> • Add new dimension: To extend a cube with a new extension, create a new dimension, or use a pre-delivered SAP dimension and add it as an association to the corresponding dimension. • Add new field (measure): Extend the original cube with additional measures for new calculations.
Analytical Query Extensions Add new numeric fields to the query to extend the scope of the data analysis.	Extensibility Enablement Add annotations and extension include structures to the original data model to enable data model extensions, like additional numeric fields for calculations in queries.
	Extension Development Add new fields to extend the scope of the data analysis.
Service Definition Extension Create a new UI based on a released query or extend the service definition to add new queries to a service definition.	Extensibility Enablement Enable the service definition for extensibility to add additional queries to a service.
	Extension Development <ul style="list-style-type: none"> • Create a custom UI based on a released UI to adapt the UI to your business requirements. • Add a new query to a service definition to extend the original service definition with additional queries.

3.2.4 Develop with Asynchronous Programming Patterns

Asynchronous communication enables applications to execute operations independently, promoting efficiency and performance.

Asynchronous communication is particularly beneficial in complex, data-intensive apps where certain tasks can be completed in parallel, reducing overall execution time.

In ABAP Cloud, both synchronous and asynchronous patterns are supported.

Developing Synchronous and Asynchronous Process Integration

Process integration can be implemented synchronously or asynchronously. With a synchronous process integration, the data exchanging data is done locally or remotely in real time. With this point-to-point integration pattern, the sender system waits for a response from the consumer before triggering any follow-up actions. Asynchronous integration in contrast allows you to execute tasks concurrently and independently of

the main program flow. It allows for the parallel execution of time-consuming or resource-intensive operations, and response times. A typical example for an implementation of this publisher-subscriber pattern is event-based integration with local or remote business events.

To decide which pattern to use, you've to take the following into account:

- **Loose or Tight Coupling:** Loosely coupled processes can be implemented with asynchronous process integration. If tight coupling is required, synchronous process integration is the better option.
- **Performance:** If you've performance intensive use cases, using asynchronous processes may improve performance.

Asynchronous Overview

📘 Note

Asynchronous process integration is only enabled for the transactional programming model aspect (RAP).

Technology	Description	Consumption	Further Information
Background Processing Framework	The background processing framework (bgPF) is a framework that asynchronously and reliably executes methods of applications that develop background processes. This framework should be used if a tighter coupling between publisher and consumer is required. The publisher knows its subscribers and directly calls it. That's why the bgPF offers features to control the consumption by the subscriber.	On-Stack	For more information, see Background Processing Framework .
(Remote) Business Events	Remote business events use the SAP Event Mesh as a message broker to enable process integration between different products.	Side-by-side	For more information, see Business Events .
(Local) Business Events	Events can be consumed on-stack within the same system. Local Business Events rely on an event handler class to implement the asynchronous communication.	On-Stack	For more information, see Business Event Consumption .

3.2.5 Develop Localization and Internationalization

This chapter explains how apps and services can be translated and adapted to country/region-specific requirements.

About Localization and Internationalization

Localization and internationalization aim to adapt software for different languages, regional differences, and technical requirements of a target locale. Internationalization, often abbreviated as i18n, is the process of designing software so that it can be easily adapted to various languages and regions without engineering changes. Localization is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translated text.

Internationalize Apps and Services

With internationalization, you can adapt your solutions to a country/region-specific scope:

- Create country/region-specific business configuration (for example currency conversion) and create country/region-specific business configuration apps. For more information about configuration apps, see [Business Configuration Maintenance Object](#).
- Create country/region-specific extensibility (for example custom fields on business objects with country/region-specific information).

Localize Apps and Services

With the `Maintain Translations` app, you can localize custom development objects for developer extensibility, key user extensibility, or configuration content.

Localize Key User Objects

With key user apps, you can extend SAP Fiori apps with custom fields, or create other extensibility items. Each key user extension is translatable and can be localized in different languages:

Translation Solution	Translation Scope	More Information
Maintain Translations	Translates all text sources for key user extensions such as custom fields and more.	Maintain Translations

Translation Solution	Translation Scope	More Information
Translation in App	Translation workflow is included in the specific app. The translation scope in this case is always app-specific, for example in <i>Custom Reusable Elements</i> , you can only translate text from this app.	<ul style="list-style-type: none"> • Translating Custom Business Objects • Custom Fields - Uploading Code Lists with Translations • Custom Reusable Elements - Uploading Custom Code Lists and Translations
Maintain Customizing Translations	Translates customizing content such as, for example, SAP Fiori Launchpad pages.	Maintain Customizing Translations

Localize Development Objects

You can extend RAP BOs, analytical data models, or other SAP content. Each custom development or extension is translatable:

Translation Solution	Translation Scope	More Information
Maintain Translations	Translates all text sources for use cases and business configurations. This includes translation for the analytical and transactional use cases.	Maintain Translations

3.2.6 Document Apps and Services

This chapter explains how to document apps and services.

About Documenting Source Code and Development Objects

Source code and development object documentation explains the structure, functionality, and logic of the code, enabling both the original developers and newcomers to understand the software more effectively, ensuring supportability and code quality. In ABAP Cloud, you can use ABAPDoc and Knowledge Transfer Documents (KTD) to document source code and development objects for all programming aspects.

Document Source Code with ABAPDoc

ABAPDoc is a comment-based approach to documenting source code, mainly the code for classes, and interfaces. With ABAP Doc, you can document parameters, as well as exceptions, directly in the source code. For more information about ABAP Doc, see [ABAP Doc \(ABAP- Keyword Documentation\)](#).

Document Development Object with Knowledge Transfer Documents

Knowledge Transfer Documents (KTD) is a mark-down based documentation tool that allows you to document development objects such as CDS views or RAP business objects, as well as extensions. For more information about KTD, see [Knowledge Transfer Documents](#).

3.3 Test

This chapter explains the requirements and considerations during the test phase.

Prerequisites

Before testing, complete the steps to make a Fiori app available on the launchpad as explained in [Make the App Available on the Launchpad \[page 53\]](#). For integration services, create a communication scenario as outlined in [Create Communication Arrangement \(Administrator\) \[page 69\]](#).

Developing a Sustainable Test Strategy

Automated software testing ensures software quality. Automated tests play a vital role in ensuring the reliability, stability, and quality of software by quickly identifying bugs and regressions. These tests enable you to make changes and add new features while knowing that existing functionality remains intact. ABAP CCloud supports several levels of functional and automated tests, such as unit and integration tests to guarantee end-to-end quality assurance for your development process.

ABAP Cloud supports test-driven development and allows you to test your code for functional correctness in all stages of the development process. ABAP Cloud offers built-in testing capabilities with the ABAP Test Cockpit and ABAP Unit to create tests for data models, behavior, or business events in the different programming aspects.

For more information about ABAP Cloud and test-driven development with , see [Test-Driven Development with ABAP Unit](#) .

Developing Isolated Unit and Integration Tests

Isolated tests ensure that errors are reproducible, and the reasons for errors or regressions can be easily identified. Isolation of unit and integration tests guarantees that only one specific aspect of a component or

an implementation is tested, avoiding side effects in the test setup. ABAP Cloud offers multiple test double frameworks as part of the ABAP development tools for Eclipse:

Test Double Framework Overview

Test-Double Framework	Description	Object under Test	Depends on	Double	Relevant Use Case
SQL Test Double Framework	This framework enables you to manage database dependencies by creating test doubles for the database tables. The test uses these test doubles instead of the real database tables.	ABAP SQL Code	<ul style="list-style-type: none"> Database Table CDS View 	Database Table	
CDS Test Double Framework	The framework handles database dependencies. In contrast to the SQL Test Double Framework, you use the CDS Test Double Framework when the code that's being tested is designed to access code through CDS entities. The CDS Test Double Framework then redirects entity calls to doubled database tables.	CDS View	<ul style="list-style-type: none"> Database Table CDS View 	Database Table	<ul style="list-style-type: none"> Analytical Programming Model Aspect Transnational Programming Model Aspect (Data Model) Integration Programming Model Aspect

Test-Double Framework	Description	Object under Test	Depends on	Double	Relevant Use Case
RAP BO Test Double Framework	This framework is used to isolate dependencies on RAP business objects that result from EML statements. Such a dependency can be, for example, a managed read/modify operation and/or a determination that is triggered by such an operation. The RAP BO Test Double Framework isolates the dependencies by replacing the actual provider implementations with mocked implementations. The RAP BO Test Double Framework provides these mocked implementations and calls them when respective EML statements are run.	RAP BO			<ul style="list-style-type: none"> Transactional Use Case (Behavior)
RAP Event Test Double Framework	This framework is used to test whether Business Events implemented in RAP behavior pools are raised as expected.	Business Event	RAP BO		<ul style="list-style-type: none"> Transactional Use Case (Events)
ABAP OO Test Double Framework	This framework is used to manage dependencies in ABAP Objects.	ABAP Code	Object	Object	Integration Use Case

Test-Double Framework	Description	Object under Test	Depends on	Double	Relevant Use Case
Test Seams	This framework manages dependencies using test seams, a technique that protects the stability and independence of tests from any changes made in dependent components.	ABAP Code	Authority Object	Statement	

Ensuring Continuous Quality Assurance

In addition to automated tests, the ABAP Test Cockpit (ATC) provides continuous quality assurance. ATC not only supports automated tests, but it also helps you maintain and measure the quality of your software. You can evaluate your development objects for a wide range of quality issues. This includes checking for syntax errors, potential performance issues, less efficient or possibly flawed programming, compliance with standards, errors in ABAP Unit testing, and much more.

For more information about the ATC, see [Checking Quality of ABAP Code with ATC](#).

Testing the UI

To test the UI, SAPUI5 provides several testing options, such as unit and integration tests and the OData V2 mock server. For more information, see [Testing \(SAPUI5\)](#).

What's Next

Once you've developed tests, you can deploy your developments across the landscape, see [Deploy \[page 88\]](#).

[Develop Tests \[page 87\]](#)

This chapter explains how to develop unit and integration tests for the different use cases.

3.3.1 Develop Tests

This chapter explains how to develop unit and integration tests for the different use cases.

About ABAP Unit

ABAP unit is the state-of-the-art unit testing framework for ABAP. It's embedded into the ABAP programming language, which supports you in writing unit tests. In ADT, you have various options for running unit tests, and to evaluate the results around functional correctness and code coverage. With mock-frameworks to mimic dependencies, for example, for RAP BOs, CDS views, SQL, or business events, you can write automated unit tests that ensure the correctness of your code. For more information about how to use ABAP unit and the available test-double frameworks, see [Unit Testing with ABAP Unit](#).

Develop Tests for Transactional Use Cases

Developing Unit Test for the Transactional Services

You can use the CDS TDF to implement unit tests for the data model of transactional Services. With the RAP BO TDF, you can manage the dependencies in the RAP BO behavior implementation and develop unit tests for RAP BO behavior, such as actions or determinations.

For an example, see:

- [Developing Unit Tests for a CDS View](#)
- [Developing Unit Tests for a Behavior Implementation](#)

Develop Integration Tests for Transactional Services

With the OData Client Proxy and the RAP BO TDF, you can implement integration tests to test use cases through EML and OData, where multiple functional units are involved. Reading or creating instances from outside, for example, use the whole RAP application and involves dependent operations, such as determinations and validations. The integration tests validate whether the interaction between the involved functional units works as expected.

For an example, see:

- [EML Integration Tests](#)
- [OData Integration Tests](#)

For more information about the OData Client Proxy, see [OData Client Proxy-Consumption Types](#).

Develop Tests for Analytical Use Cases

You can use the CDS test double framework to test the multidimensional data models, such as dimensions and cubes. For an overview of test double frameworks, see [Test \[page 83\]](#).

Develop Tests for Integration Use Cases

You can use the available test-double frameworks to test the integration services. For an overview of test double frameworks, see [Test \[page 83\]](#).

3.4 Deploy

This chapter explains how to deploy and transport your app or service with ABAP lifecycle management.

Transporting with Proven ABAP Lifecycle Management

ABAP Cloud is deeply integrated with the proven ABAP lifecycle management and ecosystem. Therefore, all advantages that stem from the proven assets of this approach continue to be available with ABAP Cloud. Also, ABAP Cloud supports CI/CD support and git integration, as well as a dedicated transport environment for key users on the SAP Fiori launchpad.

Different SAP products rely on different lifecycle management solutions:

Product	More Information
SAP S/4HANA Cloud	<ul style="list-style-type: none">• Transport Management
SAP BTP, ABAP environment	<ul style="list-style-type: none">• ABAP Lifecycle Management• For Add-On Development: Developing and Operating SaaS Applications
SAP S/4HANA Cloud, private edition / SAP S/4HANA	<ul style="list-style-type: none">• Change and Transport System

What's Next

Once you've deployed your app or service, you ensure that everything continues to run smoothly: [Administrate, Configure, and Monitor \[page 89\]](#).

3.5 Administrate, Configure, and Monitor

This chapter explains the requirements and tools involved with, for example, monitoring an app or service that you've developed.

Administration, Operations, and Support in ABAP Cloud

In the operate phase, the app or service is actively used in a live environment, and it's maintained to ensure its smooth and efficient functioning. This stage involves monitoring the software, fixing bugs, improving system performance, and making necessary updates or modifications based on user feedback or changes in business requirements. Most tasks belonging to system administration or operations are done by the administrator. Tasks for support and monitoring are usually carried out by the developer.

Monitor Apps and Services

Monitoring tools in ABAP Cloud allow you to track and analyze the performance of an app or service. With continuous monitoring, you can identify potential issues and bottlenecks that may affect the quality, efficiency, or functionality of the software.

Monitoring Transactional Apps and Services

Monitoring transactional apps and services is important for identifying runtime issues. The **ABAP Cross Trace** provides insights into running the RAP runtime, including the processing of OData requests, for example, in SAP Fiori apps.

The ABAP Cross Trace helps you to troubleshoot issues by providing detailed trace information on how app code is run, and by interacting with the frameworks used in RAP, for example, the ABAP Runtime.

For more information, see:

- [ABAP Cross Trace](#)
- [Working with the ABAP Cross Trace](#)

Analyzing SQL Statements

When you develop apps or services, you can use the SQL trace analysis to analyze the performance of SQL statements. With the SQL trace analysis, you can clarify whether the SQL activity of your app contributes significantly to the processing time in the system, and find out where your code causes performance issues.

For more information, see [Analyzing SQL Statements](#).

Identity and Access Management

The Identity and Access Management apps secure the access to business apps and services for your business and technical users.

For more information, see:

- **SAP S/4HANA Cloud:** [Identity and Access Management/ Identity and Access Management \(IAM\) Guide](#)
- **SAP BTP, ABAP environment:** [Identity and Access Management](#)

Communication Management

Communication management apps enable the integration of different solutions and enable data exchange between the integrated solutions and processes.

For more information, see:

- **SAP S/4HANA Cloud:** [Communication Management](#)
- **SAP BTP, ABAP environment:** [Communication Management](#)

Security

Security apps are designed to manage and ensure the security of SAP systems. They facilitate the enforcement of authorization concepts, authentication procedures, secure network communications, and data protection and privacy.

For more information, see:

- **SAP S/4HANA Cloud:** [Protect your S/4HANA Cloud](#)
- **SAP BTP, ABAP environment:** [Security](#)
- **SAP S/4HANA / SAP S/4HANA Cloud, private edition:** [Security Guide](#)

Transport Management

Transport management apps enable you to deploy and transport your app or service with ABAP lifecycle management.



For more information, see [Deploy \[page 88\]](#).

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
 - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
 - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

© 2024 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.