Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2024, 11(3):136-142

Research Article ISSN: 2394 - 658X

SAP CAPM Tools and Capabilities - Part 1

Deepak Kumar

Wilmington, USA
Deepak3830@gmail.com

ABSTRACT

The SAP Cloud Application Programming Model (CAPM) is a comprehensive framework for building sophisticated, cloud-native applications within the SAP ecosystem. Grounded in domain-driven design (DDD), CAP empowers developers to model complex business domains using Core Data Services (CDS). CDS' unified, declarative syntax streamlines development by defining entities, relationships, and business logic in one place. This approach aligns applications closely with business needs, enabling rapid iteration and adaptation. CAP fosters modular, independently deployable services that seamlessly interact with each other and external systems through robust API capabilities. Leveraging Node.js for server-side logic, CAP applications excel in scalability, responsiveness, and handling asynchronous operations.

CAP combines proven open-source and SAP technologies. Its infrastructure supports Node.js and Java, offering developers flexibility. The Node.js SDK, built on Express, provides a rich ecosystem of libraries. CAP also supports Java for enterprises. SAP offers tools for both environments, including SAP Business Application Studio and Visual Studio Code. Core Data Services (CDS) is CAP's foundation, modeling both domain models and service definitions. CDS models can be deployed to databases like SAP HANA. CAP's service SDKs for Node.js and Java enable service implementation and access to SAP Business Technology Platform services like authentication and authorization.

CAP smoothly integrates with the SAP Cloud Platform and SAP S/4HANA enhancing data accessibility, security, and implementation. This collaboration maximizes investments and broadens functionalities throughout the SAP environment. Prioritizing features, for businesses, adaptability and expandability CAP enables companies to develop adaptable scalable applications that fulfill contemporary business needs and foster innovation moving forward.

Keywords: SAP CAP, SAP BTP, SAP Fiori, Node.js, Java, SAP S4 HANA

INTRODUCTION

SAP Business Technology Platform (BTP) is a cloud-based platform that allows users to develop integrate and enhance SAP applications. It provides Platform as a Service (PaaS) features such, as analytics, artificial intelligence/machine learning (AI/ML), and Internet of Things (IoT) facilitating development and growth opportunities.

SAP Fiori provides user interfaces, for SAP software. Its user-friendly and adaptable layout boosts efficiency, on devices by streamlining business operations.

Node.js, a JavaScript runtime built on Chrome's V8 engine, excels at scalable, real-time applications. Its non-blocking I/O and event-driven architecture optimize server-side programming.

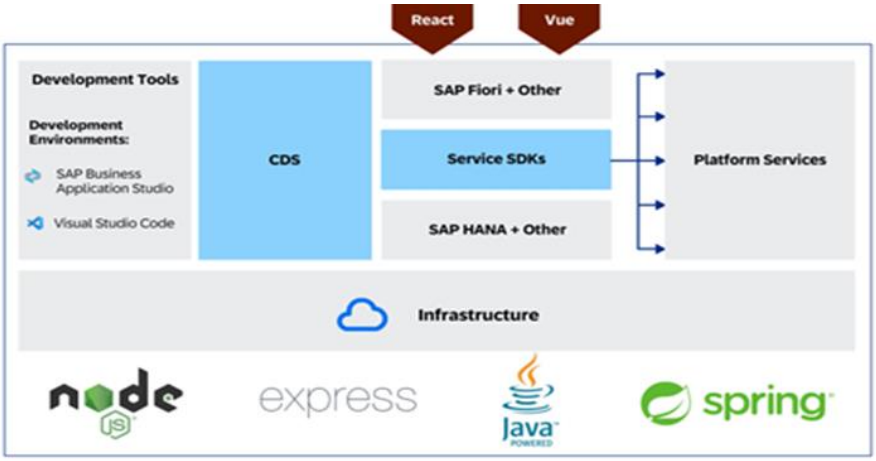
Java is an object-oriented programming language recognized for its adaptability and dedicated user base. It drives business applications on platforms. Plays a crucial role, in numerous backend operations.

SAP HANA is an in-memory database and application platform that accelerates data processing and analytics. Offering real-time insights, predictive analytics, spatial processing, and machine learning, it's a core technology for modern enterprises.

KEY FEATURES OF SAP CAPM

SAP Cloud Application Programming Model (CAP) is a comprehensive framework designed to accelerate the development of modern enterprise applications. Its core features include:

Data Modeling with Core Data Services (CDS): CAP utilizes CDS for declarative data modeling, defining entities, associations, and constraints. This streamlined approach ensures data consistency and simplifies maintenance.



Service Definition and API Exposure: CAP empowers developers to create services encapsulating business logic and expose them as APIs. This enables seamless integration with frontends, external systems, and microservices architectures.

Business Logic Implementation: Built on Node.js, CAP supports robust business logic implementation through custom JavaScript or TypeScript code. This allows for complex workflows, calculations, and event-driven processes.

Deployment and Scalability: Optimized for cloud environments, CAP supports containerized deployment on platforms like Cloud Foundry and Kubernetes. This enables scalability, resilience, and efficient resource utilization. CI/CD integration streamlines deployment processes.

By providing a unified platform for data management, service creation, business logic implementation, and deployment, CAP empowers organizations to build agile, scalable, and secure applications that meet the demands of today's dynamic business landscape.

BENEFITS OF SAP CAPM

SAP Cloud Application Programming Model (CAP) offers significant advantages that streamline development, ensure consistency, and integrate seamlessly with SAP technologies.

CAP enhances productivity by simplifying complex tasks like data modeling and service definition through its declarative Core Data Services (CDS). By eliminating boilerplate code, developers can focus on core business logic, accelerating development and reducing errors. CAP provides user-friendly database interfaces without sacrificing native capabilities, allowing full utilization of powerful platforms like SAP HANA.

Furthermore CAP emphasizes the importance of maintaining consistency by following domain driven design (DDD) principles and standardized patterns. This approach guarantees that data structures, service definitions and

APIs remain consistent across modules and projects which ultimately enhances the quality of code its maintainability and promotes collaboration, within the team.

Moreover CAP seamlessly integrates with SAP Cloud Platform and SAP S/4HANA allowing organizations to make the most out of their existing investments. By enabling integration with SAP services and extending SAP applications this integration streamlines data synchronization processes. Aligns well with enterprise IT strategies. This in turn supports the development of applications that meet enterprise grade standards.

Additionally CAP offers flexibility by supporting deployments on Cloud Foundry and Kubernetes platforms. This flexibility enables applications to easily scale up or down based on evolving business needs. Alongside extensibility options CAP provides a framework that caters to a variety of enterprise requirements while ensuring stability and high performance.

These attributes position SAP CAP as a choice, for organizations looking to develop scalable applications that are closely aligned with their business objectives and IT infrastructure.

DOMAIN DRIVEN DESIGN

CAP embraces domain-driven design (DDD), aligning applications with real-world business concepts. By focusing on core business domains like customers, orders, and products, CAP applications become more intuitive and aligned with stakeholder needs.

Core Data Services (CDS) facilitates DDD implementation through declarative data modeling. CDS defines entities, relationships, and business rules, simplifying development and ensuring consistency. CAP also supports bounded contexts, isolating application components for better management and evolution.

This DDD-centric approach results in applications that are not only technically sound but also closely aligned with business objectives. By understanding and modeling the business domain, CAP applications effectively address complex requirements and adapt to changing market conditions.

INTRODUCTION TO CORE DATA SERVICES (CDS)

Core Data Services (CDS) is a language used in SAP applications to define and manage data models. It offers a way to describe data structures, connections, and functionalities in an easy-to-understand manner. CDS plays a role, in the SAP Cloud Application Programming Model (CAP). Supports various aspects of application development from shaping data models to defining services.

In SAP CAP CDS is widely employed for setting up data models and services providing an approach to development. CDS views are instrumental in creating models that support reporting and business intelligence applications within the SAP environment. CDS aids in integrating data across SAP systems and external sources enabling data federation and harmonization within complex enterprise setups.

Within CAP CDS serves as the foundation for defining, extending and interacting with data and services. Therefore it plays a role in the learning process well. Primarily focusing on data modeling CDS enables developers to outline data structures, their interconnections, and other relevant details to offer a view of the underlying data model. Apart, from defining data models CDS also allows for service definition. This empowers developers to specify how services access and process data within CDS definitions.

According to CDS explanations CAP has the capability to automatically create components of the application, such, as database structures, OData services and even segments of the applications logic. This greatly accelerates the development timeline. These tools offer a way to represent data models and service models in a manner.

key features of CDS:

Declarative Syntax: CDS uses a syntax, to SQL making it easy for developers to define entities, attributes and relationships. This approach reduces the need for code. Boosts developer efficiency by offering a clear view of data models.

Domain-Specific Language: In terms of language CDS provides elements tailored for business applications allowing developers to articulate intricate business logic directly in the data model. This alignment with business terms ensures that the data model accurately mirrors the context and needs.

Data Modelling: When it comes to data modeling CDS enables developers to create entities with attributes and establish relationships between them such as associations and compositions. This feature supports the development of organized data models for effective enterprise data organization.

Service Definition: Beyond data modeling, CDS empowers developers to define services that facilitate CRUD operations (Create, Read, Update, Delete) on designated entities. By combining data modeling and service definition in one language it simplifies service development. Maintains consistency across different application layers.

Integration with SAP Ecosystem: Moreover CDS seamlessly integrates with SAP technologies, like SAP HANA, SAP S/4HANA, and SAP Cloud Platform. This integration helps make data access more efficient and supports analytics situations. Allows different SAP applications and services to work together seamlessly.

Types of CDS files:

Entity Data Models (EDM): When it comes to Entity Data Models (EDM) these models define data entities along, with their attributes and relationships. These entities essentially represent real world business objects like customers, products, or sales orders. By adding metadata annotations we can enrich the entity definitions with information such as constraints, UI hints, and semantic annotations enhancing their usability and significance.

Service Definitions: Service Definitions are all about defining services that enable data operations (Create, Read, Update, Delete. CRUD) on entities. These services offer interfaces for interacting with data making it easier to integrate and ensure interoperability. They outline how entities can be accessed and manipulated through service operations while adhering to service-oriented architecture (SOA) principles for a scalable application design.

Query Definitions: Moving on to Query Definitions. They define data projections and queries that wrap up data selections, aggregations and transformations. CDS views simplify the retrieval and manipulation of data ultimately boosting performance and supporting scenarios.

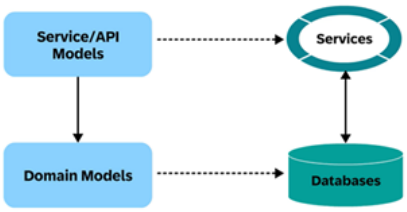
Behavior Definitions: Behaviour Definitions come into play by specifying aspects like validations, authorizations, and actions that govern how entities are accessed and manipulated. These behavioral aspects enforce business rules while enhancing data integrity within applications.

Annotations and Metadata: Lastly Annotations and Metadata play a role by enhancing CDS definitions with metadata that describes the characteristics and behavior of entities as well as attributes and relationships. By providing meaning to data elements these annotations improve the understanding of data while promoting integration, across layers of the application architecture.

Domain models:

Domain models outline the business elements, their characteristics and how they are interconnected. They define the data structures of specific service implementations. These models form the foundation, for storing data in databases. Also serves as the framework for services that act as interfaces for accessing data. Service or API models describe how external users or front-end applications can interact with interfaces, functions, and endpoints. Services are responsible for executing the business logic by processing and converting data based on the definitions in these models. They. Present data from databases according to domain model specifications outlined in service/API models, enabling systems or applications to engage with the underlying data.

CDS models are represented using Core Schema Notation (CSN) which is akin, to JSON Schema but offers extensive capabilities beyond JSON by capturing comprehensive entity relationship models and extensions. All steps involving model processing and compilation operate on CSN objects. CSN-marked designs can be put together in end formats like OData/EDM interfaces or storage models, for SQL databases (SQL DDL statements). CSN-marked designs can be generated from origins. They can be extracted from or. yaml files. Produced on the fly, in code during execution.



In CDS files models are described using CDS Definition Language (CDL) and CDS Query Language (CQL). CDL is a way for humans to define models clearly while CQL is, like a version of SQL for writing queries. CAP parses the information, in these CDS files into CSN. An instance of a Domain Model created with CDL serves as an example.

```
namespace com.app.test;

entity Test {
    key ID    : UUID;
        LANGU : String(2);
        Date  : Date;
}
```

Understanding Keywords Used with CDS.

Definition Language: CDL (Definition Language) serves as a syntax, in SAP applications within the SAP Cloud Application Programming Model (CAP) to outline data models and services in a user-friendly manner. It simplifies the description of entities, relationships, and business logic.

Key Features; Entities and Relationships; CDL enables developers to define entities with attributes and establish relationships between them through associations and compositions.
Annotations; These provide metadata for entities and properties offering details such as constraints or UI hints.
Service Definition; CDL also supports defining services that allow CRUD operations on entities supporting service-oriented architectures (SOA) and services in a user-friendly manner. It simplifies the description of entities, relationships, and business logic.
Schema Notation: CSN (Schema Notation) is the format for representing CDS models as JavaScript objects similar to JSON Schema. It facilitates the serialization and deserialization of CDS definitions.
Query Language: CQL (CDS Query Language) expands on the SQL SELECT statement to query data models defined in CDS effectively. It includes features tailored for handling CDS elements like associations and projections.
Query Notation: CQN (Core Query Notation) is a format represented as plain JavaScript objects, used to capture queries against CDS models. It provides a standardized way to express queries programmatically, facilitating dynamic query generation and execution.
Expressions (CXN): CXN (Core Expression Notation) is used to capture expressions as plain JavaScript objects within CDS. It supports defining conditions, calculations, and transformations that can be applied within CDS definitions and queries.

Domain entities:
Domain entities are structured types that contain named and typed elements, representing sets of data within an application. These entities act as the building blocks for creating data models in CAP allowing developers to structure and handle data efficiently. When transformed into persistence models domain entities usually correspond to database tables with each entity outlining a schema that dictates how data is stored and accessed. Domain entities support CRUD (Create, Read, Update, Delete) operations enabling applications to carry out data management tasks. These operations are. Made accessible through services defined in CDS facilitating integration and interaction, with the underlying data. In CDS when entities are defined they are linked to persistence models. Typically database tables in databases or collections in NoSQL databases. CDS offers tools to define how entities are stored and managed including entity relationships (such as associations and compositions) and constraints (, like keys and indexes).

```
entity Authors {
    key ID          : UUID;
        name        : String(100);
        dateOfBirth : Date;
        dateOfDeath : Date;
}
```

Entity names should be capitalized to distinguish them clearly from other elements in the application. For example, Authors instead of authors. Entity names are recommended to be in plural form (e.g., Authors, Products) to reflect collections of similar entities. Elements within entities should start with a lowercase letter (e.g., name, description). This convention helps maintain consistency and readability within the data model definitions. Entities offer a method, for handling and arranging data in applications improving organization and ease of maintenance. By following naming guidelines and established methods domain entities encourage uniformity in data representation and support reusability throughout sections of the application. Entities enable connectivity, with SAP technologies and external systems fostering scalable application growth and data compatibility.

Providing data type to CDS elements:
Build in Types in CDS elements:

Type	description	SQL equivalent
UUID	An opaque string one such example 'be071623-8699-4106-...	NVARCHAR (36).
Boolean	Holds true or false.	BOOLEAN.
UInt8, Int16, Int32 and Int64	Integer type	TINYINT, SMALLINT, INTEGER, and BIGINT respectively.
Integer, Integer64	Integer type	INTEGER, BIGINT respectively
Decimal, Double	Floating-point value.	DECIMAL., DOUBLE respectively.\
Date and Time	Date and time object data.	DATE and TIME
DateTime and TimeStamp	DateTime object data	TIMESTAMP type in SQL.
String	String values	NVARCHAR
Binary	Binary values	VARBINARY
LargeBinary	Binary values	BLOB
LargeString	Binary values	NCLOB
Vector	Vector values	REAL_VECTOR

Custom types in CDS elements:

Type.	Description.	Example.
Simple Types	It is based on the built-in data type. Example is "type number: Integer;" now you can use the number to define a type of entity element.	`entity Books { stock : NoOfBooks; }` `type NoOfBooks : Integer;`
Structured types	These are custom data types that combine one or more related elements under a single type. They make it possible to create complex data structures and use them within entities	`entity Books { price : Price; }` `type Price { amount : Decimal; currency : String(3); }`
Enumerations:	Enumerations (or enums for short) can be used to make code more readable and self-explanatory, as they allow you to replace cryptic values with symbols in the application logic.	`entity Books { genre : Genre; }` `type Genre : Integer enum { fiction = 1; non_fiction = 2; }`

Most commonly used annotations:

Annotation	Description
@readonly	Elements annotated with @readonly, as well as calculated elements, are protected against write operations. That is, if a CREATE or UPDATE operation specifies values for such fields, these values are silently ignored.
@restrict	You can use the @restrict annotation to define authorizations on a fine-grained level. In essence, all kinds of restrictions that are based on static user roles, the request operation, and instance filters can be expressed by this annotation.
@requires	You can use the @requires annotation to control which (pseudo-)role a user requires to access a resource.
@mandatory	Elements marked with @mandatory are checked for nonempty input: null and (trimmed) empty strings are rejected.
@assert.unique	Annotate an entity with @assert.unique.<constraintName>, specifying one or more element combinations to enforce uniqueness checks on all CREATE and UPDATE operations.
@assert.target	Annotate a managed to-one association of a CDS model entity definition with the @assert.target annotation to check whether the target entity referenced by the association (the reference's target) exists.
@assert.format	Allows you to specify a regular expression string (in ECMA 262 format in CAP Node.js and java.util.regex.Pattern format in CAP Java) that all string input must match.
@assert.range	Allows you to specify [min, max] ranges for elements with ordinal types — that is, numeric or date/time types. For enum elements, true can be specified to restrict all input to the defined enum values.
@assert.notNull	Annotate a property with @assert.notNull: false to have it ignored during the generic not null check, for example if your persistence fills it automatically.
@path	The endpoint of the exposed service is constructed by its name, following some conventions (the string service is dropped and kebab-case is enforced). If you want to overwrite the path, you can add the @path annotation.
@odata.etag	The CAP runtimes support optimistic concurrency control and caching techniques using ETags. An ETag identifies a specific version of a resource found at a URL.
@cds.search	By default search is limited to the elements of type String of an entity that aren't calculated or virtual. Yet, sometimes you may want to deviate from this default and specify a different set of searchable elements, or to extend the search to associated entities.
@odata.Type	It is effective on scalar CDS types only and the value must be a valid OData (EDM) primitive type for the specified protocol version. Unknown types and non-matching facets are silently ignored. No further value constraint checks are applied.

CONCLUSION

Core Data Services (CDS), within the SAP environment transforms data modeling by providing a consistent language that defines entities, relationships, and annotations. By following conventions like capitalizing entity names (for example Authors) and using plural forms for collections CDS ensures a user representation that aligns with real-world business entities. This method not simplifies data structures but also seamlessly integrates with SAP technologies such as SAP HANA and SAP S/4HANA ensuring strong data management and interoperability across enterprise systems.

In addition to data modeling, CDS enables developers to define services that support CRUD operations on entities facilitating service-oriented architectures (SOA) for modular application design. CDS views optimize data retrieval improve query performance and enable analytics for deriving actionable insights. This efficient development process speeds up the time to market for SAP applications supported by community assistance and comprehensive documentation that promotes innovation and best practices. As companies navigate transformation CDS plays a role, in constructing adaptive data-focused solutions that enhance operational efficiency and drive strategic growth within the SAP ecosystem.

Declarations

Ethics approval and consent to participate: Not Applicable

Availability of data and materials: Not Applicable

Competing interests: Not Applicable

Funding: Not Applicable

REFERENCES

[1]. "SAP Cloud Application Programming Model | SAP Community," pages.community.sap.com. https://pages.community.sap.com/topics/cloud-application-programming

[2]. "Home | capire," cap.cloud.sap. https://cap.cloud.sap/docs/

[3]. Daniel7, "Introducing the Cloud Application Programming Model (CAP)," SAP Community, Jun. 05, 2018. https://community.sap.com/t5/technology-blogs-by-sap/introducing-the-cloud-application-programming-model-cap/ba-p/13354172
[4]. "SAP Cloud Application Programming Model | SAP Community," pages.community.sap.com. https://pages.community.sap.com/topics/cloud-application-programming
[5]. "SAP CAP: How Does It Help Enterprises in Agile Development?," www.gemini-us.com, Nov. 09, 2023. https://www.gemini-us.com/sap/sap-cap-how-does-it-help-enterprises-in-agile-development#:~:text=Additional%20Advantages
[6]. kumarsanjeev, "Part#1. SAP CDS views Demystification," SAP Community, Oct. 21, 2019. https://community.sap.com/t5/enterprise-resource-planning-blogs-by-members/part-1-sap-cds-views-demystification/ba-p/13399722
[7]. R. Glushach, "Domain-Driven Design (DDD): A Guide to Building Scalable, High-Performance Systems," Medium, Oct. 07, 2023. https://romanglushach.medium.com/domain-driven-design-ddd-a-guide-to-building-scalable-high-performance-systems-5314a7fe053c
[8]. "SAP Help Portal," help.sap.com. https://help.sap.com/docs/btp/sap-business-technology-platform/developing-business-applications-using-node-js
[9]. "SAP Help Portal," help.sap.com. https://help.sap.com/docs/bas/sap-business-application-studio/what-is-sap-business-application-studio
[10]. "SAP HANA Cloud | SAP Community," pages.community.sap.com. https://pages.community.sap.com/topics/hana/cloud