The SAP Cloud Application Programming Model (CAP) is a framework of languages, libraries, and tools for building enterprise-grade services and applications. It guides developers along a 'golden path' of proven best practices and a great wealth of out-of-the-box solutions to recurring tasks.
CAP-based projects benefit from a primary focus on domain. Instead of delving into overly technical disciplines, we focus on accelerated development and safeguarding investments in a world of rapidly changing cloud technologies.

On top of open source technologies, CAP mainly adds:
Core Data Services (CDS) as our universal modeling language for both domain models and service definitions.
Service SDKs and runtimes for Node.js and Java, offering libraries to implement and consume services as well as generic provider implementations serving many requests automatically.

That might sound like a contradiction, but isn't: While CAP certainly gives opinionated guidance, we do so without sacrificing openness and flexibility. At the end of the day, you stay in control of which tools or technologies to choose, or which architecture patterns to follow.

CAP is Opinionated in...

Higher-level concepts and APIs abstracting from and avoiding lock-ins to low-level platform features and protocols
Best Practices served out of the box with generic solutions for many recurring tasks
Out-of-the-box support for SAP Fiori and SAP HANA
Dedicated tools support provided in SAP Business Application Studio or Visual Studio Code.

CAP is Open as…
All abstractions follow a glass-box pattern that allows unrestricted access to lower-level things, if required.

You can always handle things your way in custom handlers, decide whether to adopt CQRS or Event Sourcing, for example ... while CAP simply tries to get the tedious tasks out of your way.

You can also choose other UI technologies, like Vue.js, or databases, by providing new database integrations.

CAP doesn't depend on those tools. Everything in CAP can be done using the @sap/cds-dk CLI and any editor or IDE of your choice.

Agnostic Design

Keeping pace with a rapidly changing world of cloud technologies and platforms is a major challenge when having to hardwire too many things to today's technologies, which might soon become obsolete. CAP avoids such lock-ins through higher-level concepts and APIs, which abstract low-level platform features and protocols to a large extent. In particular, this applies to things like:
Platform-specific deployment approaches and techniques
Platform-specific identity providers and authentication strategies
On-/Offboarding of tenants in SaaS solutions and tenant isolation
Synchronous protocols like REST, OData, or GraphQL
Asynchronous channels and brokers like SAP Event Mesh, MQ, or Kafka
Different database technologies including SQL and NoSQL
These abstractions allow us to quickly adapt to new emerging technologies or platforms, without affecting application code, thus safeguarding your investments.

Core Data Services (CDS)

CDS is our universal modeling language to capture static, as well as behavioral aspects of problem domains in conceptual, concise, and comprehensible ways, and hence serves as the very backbone of CAP.

Domain Models capture static aspects of problem domains as well-known entity-relationship models.
Associations capture relationships. Compositions extend that to easily model document structures.
Annotations allow enriching models with additional metadata, such as for UIs, Validations, Input Validation or Authorization.

Aspects allow to flexibly extend models in same or separate modules, packages, or projects; at design time or dynamically at runtime.
This greatly
promotes adaptability in verticalization and customization scenarios , especially in SaaS solutions.
Moreover, that fosters separation of concerns, for example to keep domain models clean and comprehensible, by factoring out technical concerns.

Querying & Views

All data access in CAP is through dynamic queries, which allows clients to request the exact information they really need. These powerful intrinsic querying capabilities are key enablers for serving requests automatically.
Note: The querying-based approach to process data is in strong contrast to Object-Relational Mapping (→ see also Related Concepts: CAP != ORM)

Core Query Language (CQL)

CQL is CDS's advanced query language. It enhances standard SQL with elements to easily query deeply nested object
graphs and document structures. For example, here's a query in CQL:

```
SELECT ID, addresses.country.name from Employees
```

... and the same in plain SQL:

```
SELECT Employees.ID, Countries.name FROM Employees
 LEFT JOIN Addresses ON Addresses.emp_ID=Employees.ID
```

LEFT JOIN Countries AS Countries ON Addresses.country_ID = Countries.ID

Queries are first-order objects – using CQN as a plain object notation – sent to local services directly, to remote services through protocols like OData or GraphQL, or to database services, which translate them to native database queries for optimized execution with late materialization.
We also use CQL in CDS to declare de-normalized views on the underlying domain model, such as in tailored service APIs.