agile-lab-dev / Data-Product-Specification Public

An open specification for data products in Data Mesh

Data Product Specification

This repository wants to define an open specification to define data products with the following principles in mind:

- Data Product as an independent unit of deployment
- Technology independence
- Extensibility

With an open specification it will be possible to create services for automatic deployment and interoperable components to build a Data Mesh platform.

Version 0.0.1

Data Product structure

The Data Product is composed by a general section with Data Product level information and four sub-structures describing components:

- **Output Ports**: representing all the different interfaces of the Data Product to expose the data to consumers
- **Workloads**: internal jobs/processes to feed the Data Product and to perform housekeeping (GDPR, regulation, audit, data quality, etc)

- **Storage Areas**: internal data storages where the Data Product is deployed, not exposed to consumers
- **Observability**: provides transparency to the data consumer about how the Data Product is currently working. This is not declarative, but exposing runtime data.

Each Data Product component trait (output ports, workloads, observabilities, etc.) will have a well-defined and fixed structure and a "specific" one to handle technology specific stuff. The fixed structure must be technology-agnostic. The first fields of teh fixed structure are more technical and linked to how the platform will handle them, while the last fields (specific excluded) are to be treated as pure metadata that will simplify the management and consumption.

General

- `ID: [String]*` the unique identifier of the Data Product. This will never change in the life of a Data Product. Constraints:
  - allowed characters are `[a-zA-Z0-9]` and `[_-]`
  - the ID is a URN of the form `urn:dmb:dp:$DPDomain:$DPName:$DPMajorVersion`
- `Name: [String]*` the name of the Data Product. This name is used also for display purposes, so it can contain all kind of characters. When used inside the Data Product ID all special characters are replaced with standard ones and spaces are replaced with dashes.
- `FullyQualifiedName: [Option[String]]` human-readable name that describes the Data Product.
- `Description: [String]` detailed description about what functional area this Data Product is representing, what purpose has and business related information.
- `Kind: [String]*` type of the entity. Since this is a Data Product the only allowed value is `dataproduct`.
- `Domain: [String]*` the identifier of the domain this Data Product belongs to.
- `Version: [String]*` this is representing the version of the Data Product. Displayed as `X.Y.Z` where X is the major version, Y is the minor version, and Z is the patch. Major version (X) is also shown in the Data Product ID and those fields (version and ID) must always be aligned with one another. We consider a Data Product as an independent unit of deployment, so if a breaking change is needed, we create a brand-new version of it by changing the major version. If we introduce a new feature (or patch) we will not create a new major version, but we can just change Y (new feature) or Z patch, thus not creating a new ID (and hence not creating a new Data Product). Constraints:
  - Major version of the Data Product is always the same as the major version of all of its components ,and it is the same version that is shown in both Data Product ID and component IDs.
- `Environment: [String]*` : logical environment where the Data Product will be deployed.
- `DataProductOwner: [String]` Data Product owner, the unique identifier of the actual user that owns, manages, and receives notifications about the Data Product. To make it technology independent it is usually the email address of the owner.
- `DataProductOwnerDisplayName [String]` : the human-readable version of `DataProductOwner`.

- `Email: [Option[String]]` point of contact between consumers and maintainers of the Data Product. It could be the owner or a distribution list, but must be reliable and responsive.
- `OwnerGroup [String]` : LDAP user/group that is owning the data product.
- `DevGroup [String]` : LDAP user/group that is in charge to develop and maintain the data product.
- `InformationSLA: [Option[String]]` describes what SLA the Data Product team is providing to answer additional information requests about the Data Product itself.
- `Status: [Option[String]]` this is an enum representing the status of this version of the Data Product. Allowed values are: `[Draft|Published|Retired]` . This is a metadata that communicates the overall status of the Data Product but is not reflected to the actual deployment status.
- `Maturity: [Option[String]]` this is an enum to let the consumer understand if it is a tactical solution or not. It is really useful during migration from Data Warehouse or Data Lake. Allowed values are: `[Tactical|Strategic]` .
- `Billing: [Option[Yaml]]` this is a free form key-value area where is possible to put information useful for resource tagging and billing.
- `Tags: [Array[Yaml]]` Tag labels at DP level (please refer to OpenMetadata https://docs.open-metadata.org/metadata-standard/schemas/types/taglabel).
- `Specific: [Yaml]` this is a custom section where we can put all the information strictly related to a specific execution environment. It can also refer to an additional file. At this level we also embed all the information to provision the general infrastructure (resource groups, networking, etc.) needed for a specific Data Product. For example if a company decides to create a ResourceGroup for each data product and have a subscription reference for each domain and environment, it will be specified at this level. Also, it is recommended to put general security here, Azure Policy or IAM policies, VPC/Vnet, Subnet. This will be filled merging data defined at common level with values defined specifically for the selected environment.

The **unique identifier** of a Data Product is the concatenation of Domain, Name and Version. So we will refer to the `DP_UK` as a URN which ends in the following way: `$DPDomain:$DPName:$DPMajorVersion` .

Output Ports

- `ID: [String]*` the unique identifier of the Output Port component. This will never change in the life of a Data Product or the component itself. Constraints:
  - allowed characters are `[a-zA-Z0-9]` and `[_-]` .
  - the ID is a URN of the form `urn:dmb:cmp:$DPDomain:$DPName:$DPMajorVersion:$OutputPortName` .
- `Name: [String]*` the name of the Output Port. This name is used also for display purposes, so it can contain all kind of characters. When used inside the Output Port ID all special characters are replaced with standard ones and spaces are replaced with dashes.
- `FullyQualifiedName: [Option[String]]` human-readable name that describes better the Output Port. It can also contain specific details (if this is a table this field could contain also

indications regarding the database and the schema).

- `Description: [String]` detailed explanation about the function and the meaning of the output port.
- `Kind: [String]*` type of the entity. Since this is an Output Port the only allowed value is `outputport` .
- `Version: [String]*` specific version of the output port. Displayed as `X.Y.Z` where X is the major version of the Data Product, Y is the minor feature and Z is the patch. Major version (X) is also shown in the component ID and those fields(version and ID) are always aligned with one another. Please note that the major version of the component *must always* correspond to the major version of the Data Product it belongs to. Constraints:
  - Major version of the Data Product is always the same as the major version of all of its components, and it is the same version that is shown in both Data Product ID and component ID.
- `InfrastructureTemplateId: [String]*` the id of the microservice responsible for provisioning the component. A microservice may be capable of provisioning several components generated from different use case templates.
- `UseCaseTemplateId: [Option[String]]*` the id of the template used in the builder to create the component. Could be empty in case the component was not created from a builder template.
- `DependsOn: [Array[String]]*` A component could depend on other components belonging to the same Data Product, for example a SQL Output port could be dependent on a Raw Output Port because it is just an external table. This is also used to define the provisioning order among components. Constraints:
  - This array will only contain IDs of other components of the same Data Product.
- `Platform: [Option[String]]` represents the vendor: Azure, GCP, AWS, CDP on AWS, etc. It is a free field, but it is useful to understand better the platform where the component will be running.
- `Technology: [Option[String]]` represents which technology is used to define the output port, like: Athena, Impala, Dremio, etc. The underlying technology is useful for the consumer to understand better how to consume the output port.
- `OutputPortType: [String]` the kind of output port: Files, SQL, Events, etc. This should be extensible with other values, like GraphQL or others.
- `CreationDate: [Optional[String]]` when this output port has been created.
- `StartDate: [Optional[String]]` the first business date present in the dataset, leave it empty for events, or we can use some standard semantic like: "-7D, -1Y".
- `ProcessDescription: [Option[String]]` what is the underlying process that contributes to generate the data exposed by this output port.
- `DataContract: [Yaml]` : In case something is going to change in this section, it represents a breaking change because the producer is breaking the contract, this will require to create a new version of the data product to keep backward compatibility
  - `Schema: [Array[Yaml]]` when it comes to describe a schema we propose to leverage OpenMetadata specification: Ref https://docs.open-metadata.org/metadata-standard/schemas/entities/table#column. Each column can have a tag array, and you

can choose between simples LabelTags, ClassificationTags or DescriptiveTags. Here an example of classification Tag https://github.com/open-metadata/OpenMetadata/blob/main/catalog-rest-service/src/main/resources/json/data/tags/piiTags.json.

- `SLA: [Yaml]` Service Level Agreement, describe the quality of data delivery and the output port in general. It represents the producer's overall promise to the consumers.
    - `IntervalOfChange: [Option[String]]` how often changes in the data are reflected.
    - `Timeliness: [Option[String]]` the skew between the time that a business fact occurs and when it becomes visibile in the data.
    - `UpTime: [Option[String]]` the percentage of port availability.
- `TermsAndConditions: [Option[String]]` If the data is usable only in specific environments.
- `Endpoint: [Option[URL]]` this is the API endpoint that self-describe the output port and provide insightful information at runtime about the physical location of the data, the protocol must be used, etc.
- `biTempBusinessTs: [Option[String]]` name of the field representing the business timestamp, as per the "bi-temporality" definition; it should match with a field in the related `Schema`
- `biTempWriteTs: [Option[String]]` name of the field representing the technical (write) timestamp, as per the "bi-temporality" definition; it should match with a field in the related `Schema`

* `DataSharingAgreement: [Yaml]` This part is covering usage, privacy, purpose, limitations and is independent by the data contract.
    - `Purpose: [Option[String]]` what is the goal of this data set.
    - `Billing: [Option[String]]` how a consumer will be charged back when it consumes this output port.
    - `Security: [Option[String]]` additional information related to security aspects, like restrictions, masking, sensibile information and privacy.
    - `IntendedUsage: [Option[String]]` any other information needed by the consumer in order to effectively consume the data, it could be related to technical stuff (e.g. extract no more than one year of data for good performances) or to business domains (e.g. this data is only useful in the marketing domains).
    - `Limitations: [Option[String]]` If any limitation is present it must be made super clear to the consumers.
    - `LifeCycle: [Option[String]]` Describe how the data will be historicized and how and when it will be deleted.
    - `Confidentiality: [Option[String]]` Describe what a consumer should do to keep the information confidential, how to process and store it. Permission to share or report it.
* `Tags: [Array[Yaml]]` Tag labels at OutputPort level, here we can have security classification for example (please refer to OpenMetadata https://docs.open-metadata.org/metadata-standard/schemas/types/taglabel).

- `SampleData: [Option[Yaml]]` provides a sample data of your Output Port (please refer to OpenMetadata specification: https://docs.open-metadata.org/metadata-standard/schemas/entities/table#tabledata).
- `SemanticLinking: [Option[Yaml]]` here we can express semantic relationships between this output port and other outputports (also coming from other domains and data products). For example, we could say that column "customerId" of our SQL Output Port references the column "id" of the SQL Output Port of the "Customer" Data Product.
- `Specific: [Yaml]` this is a custom section where we must put all the information strictly related to a specific technology or dependent from a standard/policy defined in the federated governance.

Workloads

- `ID: [String]*` the unique identifier of the Workload component. This will never change in the life of a Data Product or the component itself. Constraints:
  - allowed characters are `[a-zA-Z0-9]` and `[_-]`.
  - the ID is a URN of the form `urn:dmb:cmp:$DPDomain:$DPName:$DPMajorVersion:$WorkloadName`.
- `Name: [String]*` the name of the Workload. This name is used also for display purposes, so it can contain all kind of characters. When used inside the Workload ID all special characters are replaced with standard ones and spaces are replaced with dashes.
- `FullyQualifiedName: [Optional[String]]` human-readable name that describes better the Workload.
- `Description: [String]` detailed explanation about the purpose of the workload, what sources it's reading, what business logic is applying, etc.
- `Kind: [String]*` type of the entity. Since this is a Workload the only allowed value is `workload`.
- `Version: [String]*` specific version of the workload. Displayed as `X.Y.Z` where X is the major version of the Data Product, Y is the minor feature and Z is the patch. Major version (X) is also shown in the component ID and those fields(version and ID) are always aligned with one another. Please note that the major version of the component *must always* correspond to the major version of the Data Product it belongs to. Constraints:
  - Major version of the Data Product is always the same as the major version of all of its components, and it is the same version that is shown in both Data Product ID and component ID.
- `InfrastructureTemplateId: [String]*` the id of the microservice responsible for provisioning the component. A microservice may be capable of provisioning several components generated from different use case templates.
- `UseCaseTemplateId: [Option[String]]*` the id of the template used in the builder to create the component. Could be empty in case the component was not created from a builder template.
- `DependsOn: [Array[String]]*` A component could depend on other components belonging to the same Data Product, for example a SQL Output port could be dependent on a Raw

Output Port because it is just an external table. This is also used to define the provisioning order among components. Constraints:
  - This array will only contain IDs of other components of the same Data Product.
- `Platform: [Option[String]]` represents the vendor: Azure, GCP, AWS, CDP on AWS, etc. It is a free field, but it is useful to understand better the platform where the component will be running.
- `Technology: [Option[String]]` represents which technology is used to define the workload, like: Spark, Flink, pySpark, etc. The underlying technology is useful to understand better how the workload process data.
- `WorkloadType: [Option[String]]` explains what type of workload is: Ingestion ETL, Streaming, Internal Process, etc.
- `ConnectionType: [Option[String]]` an enum with allowed values: `[HouseKeeping|DataPipeline]`; `Housekeeping` is for all the workloads that are acting on internal data without any external dependency. `DataPipeline` instead is for workloads that are reading from outputport of other DP or external systems.
- `Tags: [Array[Yaml]]` Tag labels at Workload level (please refer to OpenMetadata https://docs.open-metadata.org/metadata-standard/schemas/types/taglabel).
- `ReadsFrom: [Array[String]]` This is filled only for `DataPipeline` workloads, and it represents the list of Output Ports or external systems that the workload uses as input. Output Ports are identified with `DP_UK:$OutputPortName`, while external systems will be defined by a URN in the form `urn:dmb:ex:$SystemName`. This filed can be elaborated more in the future and create a more semantic struct. Constraints:
  - This array will only contain Output Port IDs and/or external systems identifiers.
- `Specific: [Yaml]` this is a custom section where we can put all the information strictly related to a specific technology or dependent from a standard/policy defined in the federated governance.

Storage Area

- `ID: [String]*` the unique identifier of the Storage Area component. This will never change in the life of a Data Product or the component itself. Constraints:
  - allowed characters are `[a-zA-Z0-9]` and `[_-]`.
  - the ID is a URN of the form `urn:dmb:cmp:$DPDomain:$DPName:$DPMajorVersion:$StorageAreaName`.
- `Name: [String]*` the name of the Storage Area. This name is used also for display purposes, so it can contain all kind of characters. When used inside the Storage Area ID all special characters are replaced with standard ones and spaces are replaced with dashes.
- `FullyQualifiedName: [Optional[String]]` human-readable name that describes better the Storage Area.
- `Description: [String]` detailed explanation about the function and the meaning of this storage area,
- `Kind: [String]*` type of the entity. Since this is a Storage Area the only allowed value is `storage`.

- `Owners: [Array[String]]` It is an array of user/role/group related to LDAP/AD user. This field defines who has all permissions on this specific storage area
- `InfrastructureTemplateId: [String]*` the id of the microservice responsible for provisioning the component. A microservice may be capable of provisioning several components generated from different use case templates.
- `UseCaseTemplateId: [Option[String]]*` the id of the template used in the builder to create the component. Could be empty in case the component was not created from a builder template.
- `DependsOn: [Array[String]]*` A component could depend on other components belonging to the same Data Product, for example a SQL Output port could be dependent on a Raw Output Port because it is just an external table. This is also used to define the provisioning order among components. Constraints:
    - This array will only contain IDs of other components of the same Data Product.
- `Platform: [Option[String]]` represents the vendor: Azure, GCP, AWS, CDP on AWS, etc. It is a free field, but it is useful to understand better the platform where the component will be running.
- `Technology: [Option[String]]` represents which technology is used to define the storage area, like: S3, Kafka, Athena, etc. The underlying technology is useful to understand better how the data is internally stored.
- `StorageType: [Option[String]]` the specific type of storage: Files, SQL, Events, etc.
- `Tags: [Array[Yaml]]` Tag labels at Storage area level (please refer to OpenMetadata https://docs.open-metadata.org/metadata-standard/schemas/types/taglabel).
- `Specific: [Yaml]` this is a custom section where we can put all the information strictly related to a specific technology or dependent from a standard/policy defined in the federated governance.

Observability

Observability should be applied to each Output Port and is better to represent it as the Swagger of an API rather than something declarative like a Yaml, because it will expose runtime metrics and statistics. Anyway is good to formalize what kind of information should be included and verified at deploy time for the observability API:

- `ID: [String]*` the unique identifier of the observability API
- `Name: [String]*` the name of the observability API
- `FullyQualifiedName: [String]` human-readable that uniquely identifies an entity
- `Description: [String]` detailed explanation about what this observability is exposing
- `Endpoint: [URL]` this is the API endpoint that will expose the observability for each OutputPort
- `Completeness: [Yaml]` degree of availability of all the necessary information along the entire history
- `DataProfiling: [Yaml]` volume, distribution of volume over time, range of values, column values distribution and other statistics. Please refer to OpenMetadata to get our default

implementation https://docs.open-metadata.org/openmetadata/schemas/entities/table#tableprofile. Keep in mind that this is the kind of standard that a company need to set based on its needs.

- `Freshness: [Yaml]`
- `Availability: [Yaml]`
- `DataQuality: [Yaml]` describe data quality rules will be applied to the data, using the format you prefer.
- `Specific: [Yaml]` this is a custom section where we can put all the information strictly related to a specific technology or dependent from a standard/policy defined in the federated governance.

Extension Points

This document defines an open specification that can be customized and extended by adopters, in a way that best fits their use cases. Since this specification can be used as a contract between different modules not only to share metadata, but also to trigger operations and actions, we should highlight what are the points where customizations are encouraged, allowed, discouraged, or forbidden.

This specification has a version (that can be found at the top of this document) that identifies it in relation with older and newer versions of it. In general the version should be used to notify users of the changes between the current version and the previous ones:

- a change in the major version means that the two specification are not compatible (there are changes in names, types, and even the overall structure)
- a change in the minor version means that there are significant changes but that will not impact compatibility (e.g. a new field is added, a mandatory field is now optional)
- a change in the patch version means that there are no significant changes, but just bug fixes or small corrections (e.g. an improvement in the field description, a typo that was fixed, an improvement in the validation files)

CUE offers also a standard way to check if new versions of a schema are backwards-compatible with older versions. It is highly recommended to check for schema compatibilities when multiple and/or complex changes are introduced.

In the following sections we will list all the extensions and modifications of this specification and the impact they have on the overall contract:

Encouraged

These changes are the ones that should always be performed in order to enrich the specification. Any change of this kind should just increase the minor version number. Encouraged customizations are:

- definitions of the "specific" sections. These are fully customizable sections, where adopters have full control. These sections can be extended and modified to fit in the best way the use cases, so they can have also very complex structure (i.e. nested objects, nested arrays, validation rules, etc).

- introduction of new custom metadata fields at any level. New metadata can be very useful to better explain to consumers how the Data Product and its components are structured and how they behave.

Allowed

These changes are the ones that can be performed to better adapt the specification to the desired use cases. Any change of this kind should increase the minor version number. Allowed customizations are:

- change existing metadata mandatory fields to be optional. Always think twice before performing this kind of changes, since even if this is not a breaking change, it will reduce the enforced constraints. This kind of changes can also emerge if in your use case one metadata happens to be always valued with meaningless values just to fulfill the requirement; in this case you should try to understand why this is happening and improve data entry education before acting at this level.
- change existing metadata optional fields to be mandatory. This kind of enforcement could break compatibility of existing documents compliant to previous versions of the specification, but you could decide not to upgrade the major version since the changes to make old documents compliant are very easy to apply.
- add additional enum values (e.g. adding a new component type, or a custom value for the Maturity field). Of course new values will not be handled, and just passed as descriptive metadata to the provisioner modules.

N.B.: all the changes described above are allowed only if they do not affect reserved fields which are treated in the Forbidden customization.

Discouraged

These changes are the ones that should not be performed since they impact compatibility with older versions. Any change of this kind should **always** increase the major version number. Discouraged customizations are:

- change in the name or type of existing fields. This kind of change breaks compatibility with previous versions, and should be performed by keeping in mind that they will impact for sure all the logics based on those fields.
- moving fields as subfields of other sections (e.g. moving the "workload type" field as a subfield of a new "type" field). This is actually a specific case of the one above, and should be treated accordingly.
- deletion of existing fields. This is generally something that will impact a lot of modules that are leveraging the specification, and you must think very carefully before doing deletions. Think that you can always make a field optional, and this choice will impact the specification way less.

N.B.: all the changes described above are allowed only if they do not affect reserved fields which are treated in the Forbidden customization.

Forbidden

These changes are not allowed, and you **must** always check that your changes do not fall in this category. Forbidden changes are the ones that affect reserved fields, that should not change in any way (name, type, structure, etc). All the reserved fields are highlighted with a `*` character in the specification above, like `ID: [String]*` and `Name: [String]*`. Since these are the fields that are usually leveraged by downstream platform modules, any change could break the agreed contract between them.

Project setup

This project has an automatic GitHub action invoked every time an issue is merged into the "main" branch if a secret containing the remote web hook is set. If you fork this project you can set up your own Teams hook by defining a secret called "TEAMSWEBHOOK".

Releases

No releases published

Packages

No packages published

Contributors 6

Languages

● **CUE** 100.0%