

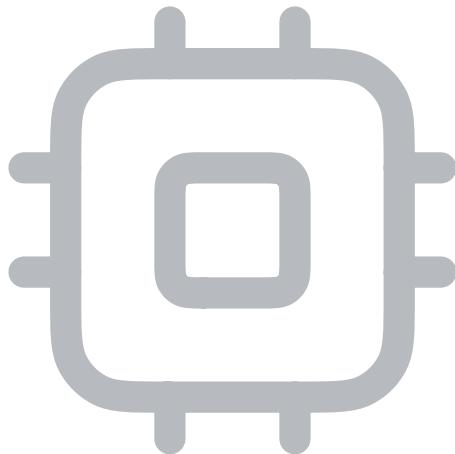
# Supergraph Modeling

Hasura believes the foundation for constructing your data supergraph should be in modeling it. Your API should be a reflection of your data, not the other way around.

By focusing on modeling your data sources and business logic, you can create a robust data layer that is easy to understand and uses a declarative approach to define data models. This empowers you to create a data layer that is highly efficient and easy to maintain.

## Quick Links

- [Learn more about the modeling your data layer.](#)



### LEARN ABOUT TYPES

Learn more about the types available to you as you model your supergraph.

[Learn about types >](#)

### CREATE MODELS

Models act as the building blocks for your supergraph. Learn how to create them.

[Create models >](#)

### DEFINE PERMISSIONS

Permissions give you a simple, declarative means of determining user roles with fine-grained access control.

[Define permissions >](#)

# Introduction

## What is metadata?

Your Hasura metadata describes every aspect of your supergraph and is used to build it into the GraphQL schema that defines your API.

This is done by validating and combining the various objects in your metadata, including subgraphs, connectors, models, commands, relationships, permissions and more.

Metadata is written in Hasura Metadata Language (HML), which is a declarative extension of YAML.

## How is metadata generated?

When you initialize a new local supergraph project, the [DDN CLI](#) scaffolds a set of metadata files and default subgraphs and configurations for you to start building your API.

The CLI is also used to perform many other metadata creation and management operations such as:

- Creating new subgraphs
- Managing the connection to data sources
- Generating models and commands from existing data sources
- Updating models and commands
- Generating relationships

## How is metadata edited?

The [Hasura VS Code extension](#) provides syntax highlighting and auto-complete for your metadata.

The DDN CLI will also validate your metadata when you use it to create a build.

## Next steps

On each page in this section, you'll find detailed information about the various components of Hasura DDN which you can author and modify in your metadata. Each page will provide you with an overview of the components, how they work, and examples of how to use them. Below

all of this, you'll find detailed reference information about the metadata structure and the fields that you can use to define each component.

# Build configs

## Introduction

Build configs are used to define the resources and other configurations required to build a **supergraph**, **subgraph** or a **connector**. They are written in YAML and are defined in individual files.

It's helpful to think of build configs as a blueprint for building your supergraph. The following options are available to you:

Config type	Description
Supergraph	Defines the configuration used to build a supergraph.
Subgraph	Defines the configuration used to build a subgraph.
Connector	Defines the configuration used to build a connector.

## How the Supergraph config works

### Lifecycle

The [Supergraph](#) object defines the configuration used to build the supergraph. While projects are generated with default configs for building for local or Hasura DDN [when initializing a new supergraph](#), you can add as many [Supergraph](#) objects as you need to define different configurations for building your supergraph.

You can then use these files to build your supergraph — either locally or on Hasura DDN — [using the CLI](#). These builds — when on Hasura DDN — can then be [applied to a project](#) which will then serve the API using the supergraph built from the configuration.

### Examples

A `supergraph.yaml` config file:

```
kind: Supergraph
version: v2
```

```

definition:
  subgraphs:
    - globals/subgraph.yaml
    - app/subgraph.yaml

```

Field	Description	Reference
<code>kind</code>	Specifies the type of configuration, in this case, <code>Supergraph</code> .	<a href="#">Supergraph</a>
<code>version</code>	The version of the <code>Supergraph</code> configuration, which is <code>v2</code> .	<a href="#">Supergraph</a>
<code>definition.subgraphs</code>	Paths to the subgraph configuration files that are included in the supergraph build.	<a href="#">SupergraphDefinitionV2</a>

## How the Subgraph config works

### Lifecycle

Each subgraph in your supergraph has its own config. The `Subgraph` object defines the configuration used to build the subgraph. While subgraphs are generated with a default config when initializing a new subgraph, you can add as many `Subgraph` objects as you need to define different configurations for building a subgraph.

You can then use these files to build a subgraph using the CLI. These subgraph builds can then be applied to a project.

### Examples

A `supergraph.yaml` config file:

```

kind: Subgraph
version: v2
definition:

```

```

name: app
generator:
  rootPath: .
includePaths:
  - metadata
envMapping:
  APP_MY_CONNECTOR_AUTHORIZATION_HEADER:
    fromEnv: APP_MY_CONNECTOR_AUTHORIZATION_HEADER
  APP_MY_CONNECTOR_READ_URL:
    fromEnv: APP_MY_CONNECTOR_READ_URL
  APP_MY_CONNECTOR_WRITE_URL:
    fromEnv: APP_MY_CONNECTOR_WRITE_URL
connectors:
  - path: connector/my_connector/connector.yaml
    connectorLinkName: my_connector

```

Field	Description	Re
kind	Specifies the type of configuration, in this case, Subgraph.	Subgraph
version	The version of the Subgraph configuration, which is v2.	Subgraph
definition.name	The name of the subgraph.	Subgraph
definition.generator.rootPath	Path to the directory containing all the subgraph metadata, in this case the current directory.	Subgraph
definition.includePaths	Paths to be included to	Subgraph

Field	Description	Ref
	construct the subgraph metadata.	
<code>definition.envMapping</code>	<p>Environment variable mapping configuration. Typically, these will correspond to connector envs.</p> <p>Additionally, when you <a href="#">initialize a connector</a>, the CLI will automatically add the required envs to the subgraph.</p>	<a href="#">EnvMapping</a>
<code>definition.connectors.path</code>	Path to the connector configuration file used in the subgraph.	<a href="#">Subgraph</a>
<code>definition.connectors.connectorLinkName</code>	Name of the connector link associated with the connector.	<a href="#">Subgraph</a>

## How the Connector config works

### Lifecycle

Each [data connector](#) in your supergraph has its own config. The [Connector](#) object defines the configuration used to build the connector. This allows you to configure the capabilities of the connector and tailor it to your needs. While connectors are initialized with default configs for

building for local or Hasura DDN, you can add as many `Connector` objects as you need to define different configurations for building your connector.

When you [initialize a connector](#), the CLI will automatically create a connector config file for you. You can then use this file to build your connector [using the CLI](#).

## Examples

A `connector.yaml` config file:

```
kind: Connector
version: v2
definition:
  name: MY_CONNECTOR
  subgraph: app
  source: hasura/postgres:v1.1.1
  context: .
  envMapping:
    CONNECTION_URI:
      fromEnv: APP_MY_CONNECTOR_CONNECTION_URI
    HASURA_SERVICE_TOKEN_SECRET:
      fromEnv: APP_MY_CONNECTOR_HASURA_SERVICE_TOKEN_SECRET
    OTEL_EXPORTER_OTLP_TRACES_ENDPOINT:
      fromEnv:
        APP_MY_CONNECTOR_OTEL_EXPORTER_OTLP_TRACES_ENDPOINT
    OTEL_SERVICE_NAME:
      fromEnv: APP_MY_CONNECTOR_OTEL_SERVICE_NAME
```

Field	Description	Reference
<code>kind</code>	Specifies the type of configuration, in this case, <code>Connector</code> .	<a href="#">Connector</a>
<code>version</code>	The version of the <code>Connector</code> configuration, which is <code>v2</code> .	<a href="#">Connector</a>
<code>definition.name</code>	The name of the connector.	<a href="#">ConnectorDefinitionV2</a>

Field	Description	Reference
<code>definition.subgraph</code>	The name of the DDN project subgraph associated with the connector.	<a href="#">ConnectorDefinitionV2</a>
<code>definition.source</code>	The versioned source of the connector.	<a href="#">ConnectorDefinitionV2</a>
<code>definition.context</code>	The path to the context directory used in the connector build.	<a href="#">ConnectorDefinitionV2</a>
<code>definition.envMapping</code>	Environment variable mapping configuration for the connector.	<a href="#">EnvMapping</a>

## Metadata structure

### Supergraph

Defines the configuration used to build the Supergraph.

Key	Value	Required	Description
<code>kind</code>	<code>Supergraph</code>	true	
<code>version</code>	<code>v2</code>	true	
<code>definition</code>	<a href="#">SupergraphDefinitionV2</a>	true	

### SupergraphDefinitionV2

Supergraph Definition V2.

Key	Value	Required	Description
subgraphs	[string]	true	Paths to subgraph configuration.

## Subgraph

Defines the configuration used to build the Subgraph.

Key	Value	Required	Description
kind	Subgraph	true	
version	v2	true	
definition	SubgraphDefinitionV2	true	

## SubgraphDefinitionV2

Subgraph Definition V2.

Key	Value	Required	Description
name	string	true	Subgraph Name.
generator	SubgraphGeneratorConfig	true	Subgraph generator Configuration.
envFile	string	false	Path to the Subgraph .env file.
includePaths	[string]	true	Paths to be included to construct Subgraph metadata.
envMapping	EnvMapping	false	Environment Variable mapping config.

Key	Value	Required	Description
connectors	[SubgraphConnector]	false	Connectors used in subgraph.

## SubgraphConnector

Subgraph Connector config.

Key	Value	Required	Description
path	string	true	Path to connector config file.
connectorLinkName	string	false	Name of connector link associated with the connector.

## EnvMapping

Environment Variables mapping config.

Key	Value	Required	Description
<customKey>	EnvSource	false	Target Environment variable.

## EnvSource

Environment Variable Source.

Key	Value	Required	Description
fromEnv	string	true	Source Environment variable.

## SubgraphGeneratorConfig

Subgraph generator Configuration.

Key	Value	Required	Description
rootPath	string	true	Path to the directory which holds all the Subgraph metadata.
graphqlRootFieldPrefix	string	false	Prefix to use while generating GraphQL root fields.
graphqlTypeNamePrefix	string	false	Prefix to use while generating GraphQL type names.
namingConvention	none / graphql	false	Naming convention to use while generating GraphQL fields and types.

## Connector

Defines the configuration used to build the connector.

Key	Value	Required	Description
kind	Connector	true	
version	v2	true	
definition	ConnectorDefinitionV2	true	

## ConnectorDefinitionV2

Connector deployment definition V2.

Key	Value	Required	Description
name	string	true	Connector name.
subgraph	string	false	DDN for subgraph name.
source	string	true	Connect Hub ID.
context	string	true	Path to the context directory
envFile	string	false	Path to the shared .env file.
envMapping	EnvMapping	false	Environment Variable mapping config.
regionConfiguration	[RegionConfigurationV2]	false	Connector deployment Region configuration

## RegionConfigurationV2

Connector deployment Region Configuration V2.

Key	Value	Required	Description
region	string	true	Region to deploy the connector to.
mode	ReadOnly / ReadWrite	true	Connector deployment mode.
envMapping	EnvMapping	false	Environment Variable mapping config.

## EnvMapping

Environment Variables mapping config.

Key	Value	Required	Description
<customKey>	EnvSource	false	Target Environment variable.

## EnvSource

Environment Variable Source.

Key	Value	Required	Description
fromEnv	string	true	Source Environment variable.

# Data Connector Links

## Introduction

A `DataConnectorLink` is used to specify the URLs and NDC schema of a [data connector](#) allowing to link it to [models](#) and [commands](#). It can be used to connect to various types of data connectors on different data sources, like SQL databases, NoSQL databases, REST APIs, GraphQL APIs, files, and more.

## How DataConnectorLinks work

### Lifecycle

A `DataConnectorLink` can be [created using the CLI](#). Out of convenience, the CLI will scaffold this file automatically for you when [initializing a new connector](#).

A `DataConnectorLink` belongs to a single [subgraph](#) and represents a connection to a data source. It is used to link the data source to the subgraph's models, commands, and relationships. The contents can be [updated using the CLI](#). This will introspect the data source and update the schema of the data connector.

Any time your data source schema changes, you should update the `DataConnectorLink` to reflect those changes. This will ensure that the schema of the data connector is up to date and that the data connector can be used to serve requests.

This configuration is then used to [generate the metadata](#) representing collections present in the data source.

#### BE MORE GRANULAR

The example we linked above is for adding all models, commands, and relationships present in a data source. However, you can add each resource individually after updating the `DataConnectorLink` configuration:

- Add models
- Add commands
- Add relationships

To make a new data connector link and it's cascading metadata available in your supergraph, you'll need to [create a new build](#) using the CLI after adding your resources.

## Examples

A sample `DataConnectorLink`:

```
kind: DataConnectorLink
version: v1
definition:
  name: data_connector
  url:
    singleUrl:
      value: http://data_connector:8100
  headers: {}
  schema:
    version: v0.1
    schema:
      scalar_types: {}
      object_types: {}
      collections: []
      functions: []
      procedures: []
  capabilities:
    version: 0.1.3
    capabilities:
      query:
        nested_fields: {}
        variables: {}
      mutation: {}
```

Field	Description
<code>kind</code>	Specifies the type of configuration. In this case, <code>DataConnectorLink</code> .
<code>version</code>	The version of the <code>DataConnectorLink</code> .

Field	Description
	configuration, which is v1.
<code>definition.name</code>	The name given to this data connector configuration.
<code>definition.url.singleUrl.value</code>	The URL used to access the data connector.
<code>definition.headers</code>	A key-value map of HTTP headers to be sent with each request to the connector.
<code>definition.schema.version</code>	The version of the schema that the data connector is using.
<code>definition.schema.schema[]</code>	The schema configuration for the data connector, representing various types, collections, functions, and procedures.
<code>definition.schema.capabilities.version</code>	The version of the capabilities that the data connector supports.
<code>definition.schema.capabilities.capabilities.query</code>	The query capabilities of the data connector.
<code>definition.schema.capabilities.capabilities.mutation</code>	The mutation capabilities of the data connector.

# Metadata structure

## DataConnectorLink

Definition of a data connector, used to bring in sources of data and connect them to OpenDD models and commands.

Key	Value	Required	Description
kind	DataConnectorLink	true	
version	v1	true	
definition	DataConnectorLinkV1	true	Definition of a data connector - version 1.

Example:

```
kind: DataConnectorLink
version: v1
definition:
  name: data_connector
  url:
    singleUrl:
      value: http://data_connector:8100
  headers: {}
  schema:
    version: v0.1
    schema:
      scalar_types: {}
      object_types: {}
      collections: []
      functions: []
      procedures: []
  capabilities:
    version: 0.1.3
    capabilities:
      query:
        nested_fields: {}
```

```
variables: {}
mutation: {}
```

## DataConnectorLinkV1

Definition of a data connector - version 1.

Key	Value	Required	Description
name	DataConnectorName	true	The name of the data connector.
url	DataConnectorUrlV1	true	The url used to access the data connector.
headers	HttpHeaders	false	Key value map of headers to be sent with each request to the connector. This is meaningful at protocol level only between engine and the data connector.

Key	Value	Required	Description
schema	SchemaAndCapabilitiesV01	true	The schema of the connector. This schema is used by the service to truthfully serve requests to the live scheduler the data connector does not log up.
argumentPresets	[DataConnectorArgumentPreset]	false	Argument presets applied to functions and processes of this connector. Default no argument preset.
responseHeaders	ResponseHeaders / null	false	HTTP response headers configuration that is forwarded from a connector to the client.

## ResponseHeaders

Configuration of what HTTP response headers should be forwarded from a data connector to the client in HTTP response.

Key	Value	Required	Description
headersField	DataConnectorColumnName	true	Name of the NDC function/procedure result which contains the response headers.
resultField	DataConnectorColumnName	true	Name of the NDC function/procedure result which contains the response.
forwardHeaders	[string]	true	List of actual response headers from the data connector to be used as response headers.

## DataConnectorColumnName

The name of a column in a data connector.

**Value:** string

## DataConnectorArgumentPreset

An argument preset that can be applied to all functions/procedures of a connector

Key	Value	Required	Description
argument	DataConnectorArgumentName	true	The name of an argument as defined by a data connector.
value	DataConnectorArgumentPresetValue	true	The value of an argument in a data connector argument preset.

## DataConnectorArgumentPresetValue

The value of a data connector argument preset.

Key	Value	Required	Description
httpHeaders	HttpHeadersPreset	true	HTTP headers that can be preset from request

## HttpHeadersPreset

Configuration of what HTTP request headers should be forwarded to a data connector.

Key	Value	Required	Description
forward	[string]	true	List of HTTP headers that should be forwarded from HTTP requests
additional	AdditionalHttpHeaders	true	Additional headers that should be forwarded, from other contexts

## AdditionalHttpHeaders

Key value map of HTTP headers to be forwarded in the headers argument of a data connector request.

Key	Value	Required	Description
<customKey>	ValueExpression	false	

## ValueExpression

An expression which evaluates to a value that can be used in permissions and various presets.

Must have exactly one of the following fields:

Key	Value	Required	Description
literal		false	
sessionVariable	string	false	

## DataConnectorArgumentName

The name of an argument as defined by a data connector.

**Value:** string

## SchemaAndCapabilitiesV01

Version 0.1 of schema and capabilities for a data connector.

Key	Value	Required	Description
version	v0.1	true	
schema	Schema Response	true	
capabilities	Capabilities Response	true	

## HttpHeaders

Key value map of HTTP headers to be sent with an HTTP request. The key is the header name and the value is a potential reference to an environment variable.

Key	Value	Required	Description
<customKey>	EnvironmentValue	false	

## DataConnectorUrlV1

A URL to access a data connector. This can be a single URL or a pair of read and write URLs.

**Must have exactly one of the following fields:**

Key	Value	Required	Description
singleUrl	EnvironmentValue	false	
readWriteUrls	ReadWriteUrls	false	A pair of URLs to access a data connector, one for reading and one for writing.

## ReadWriteUrls

A pair of URLs to access a data connector, one for reading and one for writing.

Key	Value	Required	Description
read	EnvironmentValue	true	
write	EnvironmentValue	true	

## EnvironmentValue

Either a literal string or a reference to a Hasura secret

Must have exactly one of the following fields:

Key	Value	Required	Description
value	string	false	
valueFromEnv	string	false	

## DataConnectorName

The name of a data connector.

**Value:** string

## DataConnectorScalarRepresentation

The representation of a data connector scalar in terms of Open DD types

Key	Value	Required
kind	DataConnectorScalarRepresentation	true
version	v1	true
definition	DataConnectorScalarRepresentationV1	true

## DataConnectorScalarRepresentationV1

The representation of a data connector scalar in terms of Open DD types. Deprecated in favour of [BooleanExpressionType](#).

Key	Value	F
dataConnectorName	DataConnectorName	t
dataConnectorScalarType	DataConnectorScalarType	t
representation	TypeName	t
graphql	DataConnectorScalarGraphQLConfiguration / null	f

Key	Value	F

Example:

```
dataConnectorName: data_connector
dataConnectorScalarType: varchar
representation: String
graphql:
  comparisonExpressionTypeName: String_Comparison_Exp
```

## DataConnectorScalarGraphQLConfiguration

GraphQL configuration of a data connector scalar

Key	Value	Required	Description
comparisonExpressionTypeName	GraphQLTypeName / null	false	

## GraphQLTypeName

The name of a GraphQL type.

**Value:** string

## TypeName

The name of the Open DD type that this data connector scalar type should be represented as.

One of the following values:

Value	Description
InbuiltType	An inbuilt primitive OpenDD type.
CustomTypeName	

## **CustomTypeName**

The name of a user-defined type.

**Value:** string

## **InbuiltType**

An inbuilt primitive OpenDD type.

**Value:** `ID` / `Int` / `Float` / `Boolean` / `String`

## **DataConnectorScalarType**

The name of a scalar type in a data connector.

**Value:** string

## **DataConnectorName**

The name of a data connector.

**Value:** string

# Types

## Introduction

Types serve as the fundamental elements that define the structure of your data.

Being able to define types in your data domain is beneficial because it provides you with the flexibility to define them separately from the types referenced by a data connector's source.

The specification employs a concrete type system that includes both primitive and user-defined types. All subsequent layers, such as [models](#), [commands](#), and [relationships](#) are defined in terms of these types.

The types can be one of the following:

Type	Description
Primitive	These are the basic types <code>ID</code> , <code>Int</code> , <code>Float</code> , <code>Boolean</code> , or <code>String</code> .
Custom	These are user-defined types, such as <code>ScalarType</code> or <code>ObjectType</code> .
Type References	When specifying the types of a field or an argument, you can mark them as required <code>!</code> or repeated <code>[]</code> .

The spec also allows you to map existing data connector scalars to types in your data domain.

### (!) PRIMITIVE TYPES AND TYPE REFERENCES

Primitive types supported are `ID`, `Int`, `Float`, `Boolean` and `String`.

Type references follow [GraphQL type syntax](#). Fields and arguments are nullable by default. To represent non-nullability, specify a `!` after the type name. Similarly, array fields and arguments are wrapped in `[]`.

## How types work

### Lifecycle

Typically, types will be generated from the data connector schema when you [introspect a data connector](#). You can also define custom types to represent data that doesn't exist in the data connector.

Further, you can define custom types by either aliasing existing types (such as primitives or custom), or you can define a type with fields. In turn, the fields themselves can be a primitive or another custom type.

Type references are types of fields and arguments that refer to other primitive or custom types and which can be marked as nullable, required or repeated (in the case of arrays).

To make a new types available in your supergraph, you'll need to [create a new build](#) using the CLI.

## Examples

### ScalarType

A sample scalar type definition:

```
kind: ScalarType
version: v1
definition:
  name: Uuid
  graphql:
    typeName: Uuid
```

Field	Description	Reference
kind	Indicates that this configuration is for a custom scalar type.	ScalarType
version	The version of the ScalarType configuration.	ScalarTypeV1

Field	Description	Reference
<code>definition.name</code>	The unique name for your custom scalar type, used throughout your project.	<a href="#">CustomTypeName</a>
<code>definition.graphql.typeName</code>	The name to use for this scalar type in your GraphQL schema.	<a href="#">ScalarTypeGraphQLConfig</a>
<code>definition.description</code>	An optional description of this scalar type that appears in the GraphQL schema.	<a href="#">ScalarTypeV1</a>

## ObjectType

A sample object type definition:

```

kind: ObjectType
version: v1
definition:
  name: CartItems
  fields:
    - name: cartId
      type: Uuid!
    - name: createdAt
      type: Timestamptz
    - name: id
      type: Uuid!
    - name: productId
      type: Uuid!
    - name: quantity
      type: Int4!
  
```

```

- name: updatedAt
  type: Timestamptz
graphql:
  typeName: CartItems
  inputTypeName: CartItemsInput
dataConnectorTypeMapping:
- dataConnectorName: my_pg
  dataConnectorObjectType: cart_items
  fieldMapping:
    cartId:
      column:
        name: cart_id
    createdAt:
      column:
        name: created_at
    id:
      column:
        name: id
    productId:
      column:
        name: product_id
    quantity:
      column:
        name: quantity
    updatedAt:
      column:
        name: updated_at

```

Field	Description
kind	Indicates that this configuration is for a custom object type.
version	The version of the ObjectType configuration.

Field	Description
<code>definition.name</code>	The unique name for your custom object type, used throughout your project.
<code>definition.fields[].name</code>	The name of a field within your object type.
<code>definition.fields[].type</code>	The data type for the field, which can be a primitive, custom type, or a type reference (e.g., <code>Uuid!</code> , <code>Int4!</code> ).
<code>definition.fields[].description</code>	An optional description of this field that appears in the GraphQL schema.
<code>definition.graphql.typeName</code>	The name to use for this object type in your GraphQL schema.
<code>definition.graphql.inputTypeName</code>	The name to use for this object type in input operations within your GraphQL schema.

Field	Description
<code>definition.dataConnectorTypeMapping[]</code>	The mapping of data connector object types to your object type.
<code>definition.dataConnectorTypeMapping[].fieldMapping[]</code>	The mapping of fields in your object type to columns in a data connector.

## Metadata structure

### ScalarType

Definition of a user-defined scalar type that has opaque semantics.

Key	Value	Required	Description
<code>kind</code>	<code>ScalarType</code>	true	
<code>version</code>	<code>v1</code>	true	
<code>definition</code>	<code>ScalarTypeV1</code>	true	Definition of a user-defined scalar type that has opaque semantics.

Example:

```
kind: ScalarType
version: v1
name: CustomString
graphql:
```

```
typeName: CustomString  
description: A custom string type
```

## ScalarTypeV1

Definition of a user-defined scalar type that has opaque semantics.

Key	Value	Required	Description
name	CustomTypeName	true	The name give this scalar type used to refer to it elsewhere in the metadata. Must be unique across all types defined in this subgraph.
graphql	ScalarTypeGraphQLConfiguration / null	false	Configuration for how this scalar type should appear in the GraphQL schema.
description	string / null	false	The description of this scalar. Gets added to the description of the scalar's definition in the graph schema.

## ScalarTypeGraphQLConfiguration

GraphQL configuration of an Open DD scalar type

Key	Value	Required	Description
typeName	GraphQLTypeName	true	The name of the GraphQL type to use for this scalar.

## GraphQLTypeName

The name of a GraphQL type.

**Value:** string

## CustomTypeName

The name of a user-defined type.

**Value:** string

## ObjectType

Definition of a user-defined Open DD object type.

Key	Value	Required	Description
kind	ObjectType	true	
version	v1	true	
definition	ObjectTypeV1	true	Definition of a user-defined Open DD object type.

**Example:**

```

kind: ObjectType
version: v1
definition:
  name: Author
  fields:
    - name: author_id
      type: Int!

```

```
    description: The id of the author
  - name: first_name
    type: String
      description: The first name of the author
  - name: last_name
    type: String
      description: The last name of the author
  - name: biography
    type: String
      description: AI generated biography for the author
  arguments:
    - name: ai_model
      argumentType: String!
        description: The AI model to use for generating the
biography
    description: An author of a book
  globalIdFields:
    - author_id
  graphql:
    typeName: Author
  dataConnectorTypeMapping:
    - dataConnectorName: my_db
      dataConnectorObjectType: author
      fieldMapping:
        author_id:
          column:
            name: id
    - dataConnectorName: my_vector_db
      dataConnectorObjectType: author
      fieldMapping:
        biography:
          column:
            name: biography
          argumentMapping:
            ai_model: model
```

## ObjectTypeV1

Definition of a user-defined Open DD object type.

Key	Value	Required
name	CustomTypeName	true
fields	[ObjectFieldDefinition]	true
globalIdFields	[FieldName] / null	false
graphql	ObjectTypeGraphQLConfiguration / null	false

Key	Value	Required
<code>description</code>	string / null	false
<code>dataConnectorTypeMapping</code>	[DataConnectorTypeMapping]	false

## DataConnectorTypeMapping

This defines the mapping of the fields of an object type to the corresponding columns of an object type in a data connector.

Key	Value	Required	Description
<code>dataConnectorName</code>	DataConnectorName	true	
<code>dataConnectorObjectType</code>	DataConnectorObjectType	true	
<code>fieldMapping</code>	FieldMappings	false	

## FieldMappings

Mapping of object fields to their source columns in the data connector.

Key	Value	Required	Description
<code>&lt;customKey&gt;</code>	FieldMapping	false	

## FieldMapping

Source field directly maps to some column in the data connector.

Key	Value	Required	Description
column	ColumnFieldMapping	true	The target column in a data connector object that a source field maps to.

## ColumnFieldMapping

The target column in a data connector object that a source field maps to.

Key	Value	Required	Description
name	DataConnectorColumnName	true	The name of the target column
argumentMapping	ArgumentMapping / null	false	Argument to the column field

## ArgumentMapping

Mapping of a command or model argument name to the corresponding argument name used in the data connector. The key of this object is the argument name used in the command or model and the value is the argument name used in the data connector.

Key	Value	Required	Description
<customKey>	DataConnectorArgumentName	false	

## DataConnectorArgumentName

The name of an argument as defined by a data connector.

**Value:** string

## DataConnectorColumnName

The name of a column in a data connector.

**Value:** string

## DataConnectorObjectType

The name of an object type in a data connector.

**Value:** string

## DataConnectorName

The name of a data connector.

**Value:** string

## ObjectTypeGraphQLConfiguration

GraphQL configuration of an Open DD object type.

Key	Value	Required	Description
typeName	GraphQLTypeName / null	false	The name used for GraphQL representation of this type when used in output contexts.
inputTypeName	GraphQLTypeName / null	false	The name used for GraphQL representation of this type when used in input contexts.
apolloFederation	ObjectApolloFederationConfig / null	false	Configuration for exposing to Apollo.

Key	Value	Required	Description
			federation related and directives

## ObjectApolloFederationConfig

Configuration for apollo federation related types and directives.

Key	Value	Required	Description
keys	[ApolloFederationObjectKey]	true	

## ApolloFederationObjectKey

The definition of a key for an apollo federation object.

Key	Value	Required	Description
fields	[FieldName]	true	

## GraphQLTypeName

The name of a GraphQL type.

**Value:** string

## ObjectFieldDefinition

The definition of a field in a user-defined object type.

Key	Value	Required	Description
name	FieldName	true	The name of the field. This name is used both when referring to the field elsewhere in the metadata

Key	Value	Required	Description
			and when creating the corresponding GraphQL type.
type	TypeReference	true	The type of this field. This uses the GraphQL syntax to represent field types and must refer to one of the inbuilt OpenDd types or another user-defined type.
description	string / null	false	The description of this field. Gets added to the description of the field's definition in the graphql schema.
deprecated	Deprecated / null	false	Whether this field is deprecated. If set, the deprecation status is added to the field's graphql schema.
arguments	[FieldArgumentDefinition]	false	The arguments for the field

## FieldArgumentDefinition

The definition of an argument for a field in a user-defined object type.

Key	Value	Required	Description
name	ArgumentName	true	
argumentType	TypeReference	true	
description	string / null	false	

## ArgumentName

The name of an argument.

**Value:** string

## Deprecated

OpenDd configuration to indicate whether an object type field, relationship, model root field or command root field is deprecated.

Key	Value	Required	Description
reason	string / null	false	The reason for deprecation.

## TypeReference

A reference to an Open DD type including nullable values and arrays. Suffix '!' to indicate a non-nullable reference, and wrap in '[]' to indicate an array. Eg: '[String!]!' is a non-nullable array of non-nullable strings.

**Value:** string

## FieldName

The name of a field in a user-defined object type.

**Value:** string

## CustomTypeName

The name of a user-defined type.

**Value:** string

# Models

## Introduction

In Hasura DDN, a **model** represents a collection of data objects within a data domain. Models act as the foundational elements (or "nouns") that define the data structure and behavior in your API. They can be backed by various data sources such as database tables, custom SQL queries, materialized views, or even external REST or GraphQL APIs.

Models bridge the gap between [data connectors](#) and the [GraphQL API](#), enabling operations like insertion, updating, deletion, and querying with features like filtering, pagination, and sorting.

## How models work

### Lifecycle

You can quickly add models to your metadata [using the CLI](#).

Models themselves are backed by [types](#), which are derived using the schema of the data source they represent via a [DataConnectorLink object](#). This means that when you add a model, Hasura DDN automatically generates the necessary types for you.

Once a model is declared, it becomes a central reference point within your metadata. Models are often associated with [Relationship](#) objects, allowing them to interact with other models, and with [Permissions](#) objects, which control access to the data they represent. Like all other metadata objects, models are defined in an HML file.

You should [update your models](#) whenever you make changes to your data sources and, in turn, your DataConnectorLink objects. This ensures that your API remains in sync with your data.

To make a new model available in your supergraph, you'll need to [create a new build](#) using the CLI.

### Examples

The following is an example of a model definition for a Users model:

```
---
```

```
kind: Model
version: v1
definition:
  name: Users
  objectType: Users
  source:
    dataConnectorName: my_pg
    collection: users
  filterExpressionType: UsersBoolExp
  orderableFields:
    - fieldName: id
      orderByDirections:
        enableAll: true
    - fieldName: name
      orderByDirections:
        enableAll: true
    - fieldName: email
      orderByDirections:
        enableAll: true
    - fieldName: createdAt
      orderByDirections:
        enableAll: true
  graphql:
    selectMany:
      queryRootField: users
    selectUniques:
      - queryRootField: usersById
        uniqueIdentifier:
          - id
  orderByExpressionType: UsersOrderBy
```

Field	Description	Ref
kind	Specifies the type of object being defined. In	Model

Field	Description	Ref
	this case, it's a model.	
<b>version</b>	Indicates the version of the model's structure.	<a href="#">ModelV1</a>
<b>definition.name</b>	The name of the model, representing the collection of data objects within this model.	<a href="#">modelName</a>
<b>definition.objectType</b>	Defines the type of objects contained within this model.	<a href="#">CustomTypeName</a>
<b>definition.source.dataConnectorName</b>	The name of the data connector that backs this model, linking it to the actual data source.	<a href="#">DataConnectorName</a>
<b>definition.source.collection</b>	The specific collection within the data connector that this model maps to.	<a href="#">CollectionName</a>

Field	Description	Reference
<b>definition.filterExpressionType</b>	Specifies the type used for filtering the model's data within GraphQL queries.	<a href="#">CustomTypeName</a>
<b>definition.orderableFields</b>	A list of fields (in this example: <code>id</code> , <code>name</code> , <code>email</code> , <code>createdAt</code> ) that can be used to sort the data in this model.	<a href="#">OrderableField</a>
<b>definition.graphql.selectMany.queryRootField</b>	The root field in the GraphQL API for querying multiple objects from this model. Removing this will disable the ability to query and return an array of this model.	<a href="#">SelectManyGraphQL</a>
<b>definition.graphql.selectUniques</b>	Defines unique query root fields (e.g., <code>usersById</code> ) and identifiers used to retrieve unique objects in GraphQL. Removing this will	<a href="#">SelectUniqueGraphQL</a>

Field	Description	Ref
	disable the ability to query a single instance of this model.	
<code>definition.graphql.orderByExpressionType</code>	The type name used for specifying how to order the data when querying this model in GraphQL.	<a href="#">GraphQLTypeN</a>

The earlier model definition enables the following query in the API:

```
query UsersQuery {
  users(where: { name: { _eq: "Bob" } }, order_by: { createdAt: Asc }, limit: 10) {
    id
    name
    email
    createdAt
  }
}
```

The above example works because the earlier model definition includes the necessary configuration for filtering, sorting, and pagination. Alternatively, if we'd set `enableAll` to `false` for the `createdAt` field in the `orderableFields` section, the `createdAt` field would not be available for sorting in the API.

- ▶ Check out Global IDs for relay:

# Metadata structure

## Model

The definition of a data model. A data model is a collection of objects of a particular type. Models can support one or more CRUD operations.

Key	Value	Required	Description
kind	Model	true	
version	v1	true	
definition	ModelV1	true	The definition of a data model. A data model is a collection of objects of a particular type. Models can support one or more CRUD operations.

Example:

```
kind: Model
version: v1
definition:
  name: Articles
  objectType: article
  globalIdSource: true
  arguments: []
  source:
    dataConnectorName: data_connector
    collection: articles
    argumentMapping: {}
  filterExpressionType: Article_bool_exp
  orderableFields:
    - fieldName: article_id
      orderByDirections:
        enableAll: true
    - fieldName: title
      orderByDirections:
        enableAll: true
    - fieldName: author_id
```

```

    orderByDirections:
      enableAll: true
graphql:
  selectUniques:
    - queryRootField: ArticleByID
      uniqueIdentifier:
        - article_id
      description: Description for the select unique ArticleByID
  selectMany:
    queryRootField: ArticleMany
    description: Description for the select many ArticleMany
  orderByExpressionType: Article_Order_By
  apolloFederation:
    entitySource: true
description: Description for the model Articles

```

## ModelV1

The definition of a data model. A data model is a collection of objects of a particular type. Models can support one or more CRUD operations.

Key	Value	Required	Description
name	modelName	true	The name of the data model.
objectType	CustomTypeName	true	The type of objects in the model.
globalIdSource	boolean	false	Whether or not the global ID should be generated for all objects in the type.
arguments	[ArgumentDefinition]	false	A list of arguments accepted by the model. If no arguments are present, an empty array is returned.
source	ModelSource / null	false	The source configuration for the model.

Key	Value	Required	Description
modelType	CustomTypeName / null	false	The model type of the model.
filterExpressionType	CustomTypeName / null	false	The filter expression type of the model.
orderableFields	[OrderableField]	true	A list of fields that can be ordered by the client.
aggregateExpression	AggregateExpressionName / null	false	The aggregate expression that is used in the aggregation mode.
graphql	ModelGraphQLDefinition / null	false	Configuration how the model should be represented in GraphQL.
description	string / null	false	The description of the model, which is added to the model schema.

## ModelGraphQLDefinition

The definition of how a model appears in the GraphQL API.

Key	Value	Required
selectUniques	[SelectUniqueGraphQLDefinition]	true

Key	Value	Required
selectMany	SelectManyGraphQLDefinition / null	false
argumentsInputType	GraphQLTypeName / null	false
orderByExpressionType	GraphQLTypeName / null	false
apolloFederation	ModelApolloFederationConfiguration / null	false
filterInputTypeName	GraphQLTypeName / null	false

Key	Value	Required
aggregate	ModelAggregateGraphQLDefinition / null	false

Example:

```

selectUniques:
  - queryRootField: ArticleByID
    uniqueIdentifier:
      - article_id
    description: Description for the select unique ArticleByID
selectMany:
  queryRootField: ArticleMany
  description: Description for the select many ArticleMany
orderByExpressionType: Article_Order_By
aggregate:
  queryRootField: ArticleAggregate
  description: Aggregate over Articles

```

## ModelAggregateGraphQLDefinition

The definition of the GraphQL API for aggregating over a model.

Key	Value	Required	Description
queryRootField	GraphQLFieldName	true	The name of the query root field for this API.
description	string / null	false	The description of the aggregate graphql

Key	Value	Required	Description
			definition for the mode. Gets added to the description of the aggregate root field in the mode in the graph schema.
<code>deprecated</code>	<code>Deprecated</code> / null	false	Whether this aggregate query field is deprecated. If set, the deprecation status is added to the aggregate root field's graphql schema.
<code>subscription</code>	<code>SubscriptionGraphQLDefinition</code> / null	false	Enable subscription on this aggregate root field.

## ModelApolloFederationConfiguration

Apollo Federation configuration for a model.

Key	Value	Required	Description
<code>entitySource</code>	boolean	true	Whether this model should be used as the source for fetching <code>_entity</code> for object of its type.

## GraphQLTypeName

The name of a GraphQL type.

**Value:** string

## SelectManyGraphQLDefinition

The definition of the GraphQL API for selecting rows from a model.

Key	Value	Required	Description
queryRootField	GraphQLFieldName	true	The name of the query root field for this API.
description	string / null	false	The description of the select many GraphQL definition for the model. Gets added to the description of the select many root field for the model in the GraphQL schema.
deprecated	Deprecated / null	false	Whether this select many query field is deprecated. If set, the deprecation status is added to the select many root field's GraphQL schema.
subscription	SubscriptionGraphQLDefinition / null	false	Enable subscription

Key	Value	Required	Description
			on this select make root field.

## SelectUniqueGraphQLDefinition

The definition of the GraphQL API for selecting a unique row/object from a model.

Key	Value	Required	Description
queryRootField	GraphQLFieldName	true	The name of the query root field for this API.
uniqueIdentifier	[FieldName]	true	A set of fields which can uniquely identify a row/object in the model.
description	string / null	false	The description of the select unique graphic definition for the model. Gets added to the description of the select unique field of model graphic scheme.
deprecated	Deprecated / null	false	Whether this select unique query is deprecated.

Key	Value	Required	Description
			is deprecated. If set, the deprecation status is added to the selected unique field's GraphQL schema.
<code>subscription</code>	<code>SubscriptionGraphQLDefinition / null</code>	false	Enable subscription on this select unique field.

## SubscriptionGraphQLDefinition

The definition of the GraphQL API for enabling subscription on query root fields.

Key	Value	Required	Description
<code>rootField</code>	<code>GraphQLFieldName</code>	true	The name of the subscription root field.
<code>description</code>	<code>string / null</code>	false	The description of the subscription GraphQL definition. Gets added to the description of the subscription root field in the GraphQL schema.
<code>deprecated</code>	<code>Deprecated / null</code>	false	Whether this subscription

Key	Value	Required	Description
			root field is deprecated. If set, the deprecation status is added to the subscription root field's graphql schema.
<code>pollingIntervalMs</code>	integer	false	Polling interval in milliseconds for the subscription.

## Deprecated

OpenDd configuration to indicate whether an object type field, relationship, model root field or command root field is deprecated.

Key	Value	Required	Description
<code>reason</code>	string / null	false	The reason for deprecation.

## GraphQLFieldName

The name of a GraphQL object field.

**Value:** string

## AggregateExpressionName

The name of an aggregate expression.

**Value:** string

## OrderableField

A field that can be used to order the objects in a model.

Key	Value	Required	Description
fieldName	FieldName	true	
orderByDirections	EnableAllOrSpecific	true	Enable all or specific values.

## EnableAllOrSpecific

Enable all or specific values.

Must have exactly one of the following fields:

Key	Value	Required	Description
enableAll	boolean	false	
enableSpecific	[OrderByDirection]	false	

## OrderByDirection

Value: Asc / Desc

## FieldName

The name of a field in a user-defined object type.

Value: string

## ModelSource

Description of how a model maps to a particular data connector

Key	Value	Required	Description
dataConnectorName	DataConnectorName	true	The name of the data connector backing this model.

Key	Value	Required	Description
collection	CollectionName	true	The collection in the data connector that backs this model.
argumentMapping	ArgumentMapping	false	Mapping from model argument names to data connector collection argument names.

Example:

```
dataConnectorName: data_connector
collection: articles
```

## ArgumentMapping

Mapping of a command or model argument name to the corresponding argument name used in the data connector. The key of this object is the argument name used in the command or model and the value is the argument name used in the data connector.

Key	Value	Required	Description
<customKey>	DataConnectorArgumentName	false	

## DataConnectorArgumentName

The name of an argument as defined by a data connector.

**Value:** string

## CollectionName

The collection in the data connector that backs this model.

**Value:** string

## DataConnectorName

The name of the data connector backing this model.

**Value:** string

## ArgumentDefinition

The definition of an argument for a field, command, or model.

Key	Value	Required	Description
name	ArgumentName	true	The name of an argument.
type	TypeReference	true	
description	string / null	false	

## TypeReference

A reference to an Open DD type including nullable values and arrays. Suffix '!' to indicate a non-nullable reference, and wrap in '[]' to indicate an array. Eg: '[String!]!' is a non-nullable array of non-nullable strings.

**Value:** string

## ArgumentName

The name of an argument.

**Value:** string

## CustomTypeName

The name of a user-defined type.

**Value:** string

## **ModelName**

The name of data model.

**Value:** string

# Commands

## Introduction

In Hasura DDN, a **command** represents an action that can be executed within your data domain. Commands act as the verbs or operations that modify or interact with the data managed by your [models](#). They can be triggered via GraphQL queries or mutations.

Commands have a fixed definition and are backed by [functions](#) (exposed as queries) or [procedures](#) (exposed as mutations) allowing you to add top-level methods to your API, or, add a custom field to an existing model.

Commands also allow you to execute [custom business logic](#) directly from your GraphQL API and are useful in validating, processing or enriching data, calling another API, or, for example, logging a user in.

## How commands work

### Lifecycle

Commands can be added to your metadata [using the CLI](#). The CLI will use a connector's [DataConnectorLink object](#) to determine which functions or procedures are available to be added as commands.

You should [update your commands](#) whenever you make changes to your data sources and, in turn, your DataConnectorLink objects. This ensures that your API remains in sync with your data.

As an example, if you're using the [TypeScript connector](#) for business logic and add a new function, you'll need to update your DataConnectorLink to ensure that the new function is present in the connector's configuration. Then, add the new command to your metadata.

To make a new command available in your supergraph, you'll need to [create a new build](#) using the CLI.

### Examples

*The following is an example of a command for a function:*

```
---
kind: Command
version: v1
definition:
  name: Hello
  outputType: String!
  arguments:
    - name: name
      type: String
  source:
    dataConnectorName: my_data_connector
    dataConnectorCommand:
      function: hello
  graphql:
    rootFieldName: hello
    rootFieldKind: Query
```

Field	Description	Reference
<b>kind</b>	Specifies the type of object being defined, which is a command in this context.	<a href="#">Command</a>
<b>version</b>	Indicates the version of the command's structure.	<a href="#">CommandV1</a>
<b>definition.name</b>	The name of the command, representing the action to be executed. You can configure this to whatever you wish.	<a href="#">CommandName</a>
<b>definition.outputType</b>	Defines the return type of	<a href="#">TypeReference</a>

Field	Description	Reference
	the command, specifying what kind of data is returned after execution.	
<b>definition.arguments</b>	Lists the arguments the command can take, allowing customization of the command's execution based on input parameters.	<a href="#">ArgumentDefinition</a>
<b>definition.source.dataConnectorName</b>	The name of the data connector that backs this command, linking it to the data source or function.	<a href="#">DataConnectorName</a>
<b>definition.source.dataConnectorCommand</b>	Specifies the function or procedure in the data connector's configuration file(s) that implements the command's logic.	<a href="#">DataConnectorCommand</a>
<b>definition.graphql.rootFieldName</b>	The name of the root field in the GraphQL API for invoking	<a href="#">GraphQLFieldName</a>

Field	Description	Reference
	this command.	
<code>definition.graphql.rootFieldKind</code>	Determines whether the command should be part of the Query or Mutation root in the GraphQL API.	<a href="#">GraphQLRootField</a>

The HML example above results in this query:

```
query {
  hello(name: "Hasura")
}
```

## Metadata structure

### Command

The definition of a command. A command is a user-defined operation which can take arguments and returns an output. The semantics of a command are opaque to the Open DD specification.

Key	Value	Required	Description
<code>kind</code>	<code>Command</code>	true	
<code>version</code>	<code>v1</code>	true	
<code>definition</code>	<code>CommandV1</code>	true	Definition of an OpenDD Command, which is a custom operation that can take arguments and returns an output. The semantics of

Key	Value	Required	Description
			a command are opaque to OpenDD.

Example:

```

kind: Command
version: v1
definition:
  name: get_latest_article
  outputType: commandArticle
  arguments: []
  source:
    dataConnectorName: data_connector
    dataConnectorCommand:
      function: latest_article
      argumentMapping: {}
    graphql:
      rootFieldName: getLatestArticle
      rootFieldKind: Query
    description: Get the latest article
  
```

## CommandV1

Definition of an OpenDD Command, which is a custom operation that can take arguments and returns an output. The semantics of a command are opaque to OpenDD.

Key	Value	Required	Description
<b>name</b>	<a href="#">CommandName</a>	true	The name of the command.
<b>outputType</b>	<a href="#">TypeReference</a>	true	The return type of the command.
<b>arguments</b>	<a href="#">ArgumentDefinition</a>	false	The list of arguments accepted by this command.

Key	Value	Required	Description
			Defaults to no arguments.
source	CommandSource / null	false	The source configuration for this command.
graphql	CommandGraphQLDefinition / null	false	Configuration for how this command should appear in the GraphQL schema.
description	string / null	false	The description of the command. Gets added to the description of the command's root field in the GraphQL schema.

## CommandGraphQLDefinition

The definition of how a command should appear in the GraphQL API.

Key	Value	Required	Description
rootFieldName	GraphQLFieldName	true	The name of the graphql root field to use for this command.
rootFieldKind	GraphQLRootFieldKind	true	Whether to put this command in the Query or Mutation root

Key	Value	Required	Description
			of the GraphQL API.
deprecated	Deprecated / null	false	Whether this command root field is deprecated. If set, this will be added to the graphql schema as a deprecated field.

**Example:**

```
rootFieldName: getLatestArticle
rootFieldKind: Query
```

## Deprecated

OpenDd configuration to indicate whether an object type field, relationship, model root field or command root field is deprecated.

Key	Value	Required	Description
reason	string / null	false	The reason for deprecation.

## GraphQLRootFieldKind

Whether to put this command in the Query or Mutation root of the GraphQL API.

**Value:** `Query` / `Mutation`

## GraphQLFieldName

The name of a GraphQL object field.

**Value:** string

## CommandSource

Description of how a command maps to a particular data connector

Key	Value	Required	Description
dataConnectorName	DataConnectorName	true	The name of the data connector backing the command.
dataConnectorCommand	DataConnectorCommand	true	The function in the data connector that backs the command.
argumentMapping	ArgumentMapping	false	Mapping of command or model argument names to data connector arguments.

Example:

```
dataConnectorName: data_connector
dataConnectorCommand:
  function: latest_article
  argumentMapping: {}
```

## ArgumentMapping

Mapping of a command or model argument name to the corresponding argument name used in the data connector. The key of this object is the argument name used in the command or model and the value is the argument name used in the data connector.

Key	Value	Required	Description
<customKey>	DataConnectorArgumentName	false	

## DataConnectorArgumentName

The name of an argument as defined by a data connector.

**Value:** string

## DataConnectorCommand

The function/procedure in the data connector that backs this command.

**Must have exactly one of the following fields:**

Key	Value	Required	Description
function	FunctionName	false	The name of a function backing the command.
procedure	ProcedureName	false	The name of a procedure backing the command.

## ProcedureName

The name of a procedure backing the command.

**Value:** string

## FunctionName

The name of a function backing the command.

**Value:** string

## DataConnectorName

The name of a data connector.

**Value:** string

## ArgumentDefinition

The definition of an argument for a field, command, or model.

Key	Value	Required	Description
name	ArgumentName	true	The name of an argument.
type	TypeReference	true	
description	string / null	false	

## ArgumentName

The name of an argument.

**Value:** string

## TypeReference

A reference to an Open DD type including nullable values and arrays. Suffix '!' to indicate a non-nullable reference, and wrap in '[]' to indicate an array. Eg: '[String!]!' is a non-nullable array of non-nullable strings.

**Value:** string

## CommandName

The name of a command.

**Value:** string

# Boolean Expressions

## Introduction

Hasura provides powerful tools to control filtering and selecting data. Boolean expression types let you control which filters are available for a [model](#) or [command](#). They can be used to configure filters on models, such as in the [filtering](#) section, or as the types of arguments to commands or models.

## How Boolean expressions work

### Lifecycle

There are two types of boolean expressions:

Type	Description
Scalar	This specifies how a user able to compare a scalar field.
Object	This specifies how fields of a type can be filtered.

Regardless of the type, boolean expressions are used to define how a user is able to filter data and are defined in your metadata.

## Examples

### Scalar

This specifies how a user is able to compare a scalar field. For instance, you might want to say that a user can only check if a `String` type is equals to another, or whether it is null or not. You can do that with the following metadata:

```
kind: BooleanExpressionType
version: v1
definition:
  name: String_comparison_exp_with_eq_and_is_null
  operand:
    scalar:
```

```
type: String
comparisonOperators:
  - name: equals
    argumentType: String!
dataConnectorOperatorMapping:
  - dataConnectorName: postgres
    dataConnectorScalarType: varchar
    operatorMapping:
      equals: _eq
logicalOperators:
  enable: true
isNull:
  enable: true
graphql:
  typeName: String_comparison_exp_with_eq_and_is_null
```

Note the `dataConnectorOperatorMapping`. This allows us to define what these operators mean in zero or more data connectors. Here, we want our `equals` operator to use Postgres's `_eq` operator.

This would allow us to write filters like:

```
{ "first_name": { "_is_null": true } }
```

```
{ "last_name": { "equals": "Bruce" } }
```

## Object

An object `BooleanExpressionType` is used to define how fields of a type can be filtered. Note that nothing here talks about specific data connectors - instead you can specify which `BooleanExpressionType` is used to filter each field or relationship, and then defer the mappings of individual scalar types to those `BooleanExpressionType`s.

```
kind: BooleanExpressionType
version: v2
definition:
  name: Album_bool_exp
```

```
operand:  
  object:  
    type: Album  
    comparableFields:  
      - fieldName: AlbumId  
        booleanExpressionType: Int_comparison_exp  
      - fieldName: ArtistId  
        booleanExpressionType:  
          Int_comparison_exp_with_is_null  
            - fieldName: Address  
              booleanExpressionType: Address_bool_exp  
    comparableRelationships:  
      - relationshipName: artist  
        booleanExpressionType: Artist_bool_exp  
logicalOperators:  
  enable: true  
isNull:  
  enable: true  
graphql:  
  typeName: Album_bool_exp
```

Note here that we can specify different comparison operators for `AlbumId` and `ArtistId` by using different `BooleanExpressionType`s for them. We are also able to define filtering on nested objects such as `Address`.

The above would let us write filters on `Album` types like:

```
{ "album": { "AlbumId": { "equals": 100 } } }
```

```
{ "album": { "Address": { "postcode": { "like": "N1" } } } }
```

```
{ "album": { "artist": { "name": { "equals": "Madonna" } } } }
```

# Metadata structure

## BooleanExpressionType

Definition of a type representing a boolean expression on an OpenDD type.

Key	Value	Required	Description
kind	BooleanExpressionType	true	
version	v1	true	
definition	BooleanExpressionTypeV1	true	Definition of a type representing a boolean expression on an OpenDD object type.

Example:

```
kind: BooleanExpressionType
version: v1
definition:
  name: Album_bool_exp
  operand:
    object:
      type: Album
      comparableFields:
        - fieldName: AlbumId
          booleanExpressionType: pg_Int_Comparison_exp
        - fieldName: ArtistId
          booleanExpressionType:
            pg_Int_Comparison_exp_with_is_null
              - fieldName: Address
                booleanExpressionType: Address_bool_exp
            comparableRelationships:
              - relationshipName: artist
                booleanExpressionType: Artist_bool_exp
            logicalOperators:
```

```
enable: true
isNull:
  enable: true
graphql:
  typeName: App_Album_bool_exp
```

## BooleanExpressionTypeV1

Definition of a type representing a boolean expression on an OpenDD object type.

Key	Value	Required
name	CustomTypeName	true
operand	BooleanExpressionOperand	true
logicalOperators	BooleanExpressionLogicalOperators	true
isNull	BooleanExpressionIsNull	true
graphql	BooleanExpressionTypeGraphQLConfiguration / null	false

Key	Value	Required

## BooleanExpressionTypeGraphQIConfiguration

GraphQL configuration of an OpenDD boolean expression type.

Key	Value	Required	Description
typeName	GraphQLTypeName	true	The name to use for the GraphQL type representation of this boolean expression type.

## GraphQLTypeName

The name of a GraphQL type.

**Value:** string

## BooleanExpressionsIsNull

Configuration for is\_null in boolean expressions

Key	Value	Required	Description
enable	boolean	true	

## BooleanExpressionLogicalOperators

Configuration for logical operators in boolean expressions

Key	Value	Required	Description
enable	boolean	true	

## BooleanExpressionOperand

Configuration for object or scalar boolean expression

Must have exactly one of the following fields:

Key	Value	Required	Description
object	BooleanExpressionObjectOperand	false	Definition of an object type representing a boolean expression on an OpenDD object type.
scalar	BooleanExpressionScalarOperand	false	Definition of a scalar type representing a boolean expression on an OpenDD scalar type.

## BooleanExpressionScalarOperand

Definition of a scalar type representing a boolean expression on an OpenDD scalar type.

Key	Value	Required
type	TypeName	true
comparisonOperators	[ComparisonOperator]	true

Key	Value	Required
dataConnectorOperatorMapping	[DataConnectorOperatorMapping]	true

## DataConnectorOperatorMapping

Mapping between OpenDD operator names and the names used in the data connector schema

Key	Value	Required	Description
dataConnectorName	DataConnectorName	true	Name of the data connector mapping
dataConnectorScalarType	DataConnectorScalarType	true	Name of the scalar type mapping
operatorMapping	operator_mapping	true	Mapping between OpenDD operator names and the names used in the data connector schema

Key	Value	Required	Description
			"_eq": "eq" if no explicit mapping is present.

## operator\_mapping

Mapping between OpenDD operator names and the data connector's operator names. Defaults to the same operator name (e.g. "\_eq: \_eq") if no explicit mapping is present.

Key	Value	Required	Description
<customKey>	DataConnectorOperatorName	false	The name of an operator in a data connector.

## DataConnectorOperatorName

The name of an operator in a data connector.

**Value:** string

## DataConnectorScalarType

The name of a scalar type in a data connector.

**Value:** string

## DataConnectorName

The name of a data connector.

**Value:** string

## ComparisonOperator

Definition of a comparison operator for a scalar type

Key	Value	Required	Description
name	OperatorName	true	Name you want to give the operator in OpenDD / GraphQL
argumentType	TypeReference	true	An OpenDD type

## TypeReference

A reference to an Open DD type including nullable values and arrays. Suffix '!' to indicate a non-nullable reference, and wrap in '[]' to indicate an array. Eg: '[String!]!' is a non-nullable array of non-nullable strings.

**Value:** string

## OperatorName

The name of an operator

**Value:** string

## TypeName

The OpenDD type name of the scalar type that this boolean expression applies to.

One of the following values:

Value	Description
InbuiltType	An inbuilt primitive OpenDD type.
CustomTypeName	

## InbuiltType

An inbuilt primitive OpenDD type.

**Value:** ID / Int / Float / Boolean / String

## BooleanExpressionObjectOperand

Definition of an object type representing a boolean expression on an OpenDD object type.

Key	Value
type	CustomTypeName
comparableFields	[BooleanExpressionComparableField]
comparableRelationships	[BooleanExpressionComparableRelationship]

## BooleanExpressionComparableRelationship

Definition of a relationship that can be used for a comparison

Key	Value	Required	Description
relationshipName	RelationshipName	true	The name of the relationship for comparison
booleanExpressionType	CustomTypeName / null	false	The boolean expression type to use for comparison. This is optional.

Key	Value	Required	Description
			relationships models, and defaults to the filterExpression of the mode

## RelationshipName

The name of the GraphQL relationship field.

**Value:** string

## BooleanExpressionComparableField

Comparison configuration definition for a field that can be used for a comparison

Key	Value	Required	Description
fieldName	FieldName	true	The name of the field that can be compared.
booleanExpressionType	CustomTypeName	true	The boolean expression type that can be used for comparison against the type of the field.

## FieldName

The name of a field in a user-defined object type.

**Value:** string

## CustomTypeName

The name of a user-defined type.

**Value:** string

## ObjectBooleanExpressionType

Definition of a type representing a boolean expression on an Open DD object type.

Key	Value	Required	
kind	ObjectBooleanExpressionType	true	
version	v1	true	
definition	ObjectBooleanExpressionTypeV1	true	Definition represents expression on object type favour of Boolean

**Example:**

```
kind: ObjectBooleanExpressionType
version: v1
definition:
  name: AuthorBoolExp
  objectType: Author
  dataConnectorName: my_db
  dataConnectorObjectType: author
  comparableFields:
    - fieldName: article_id
      operators:
        enableAll: true
    - fieldName: title
      operators:
        enableAll: true
    - fieldName: author_id
      operators:
        enableAll: true
graphql:
  typeName: Author_bool_exp
```

## ObjectBooleanExpressionTypeV1

Definition of a type representing a boolean expression on an Open DD object type.

Deprecated in favour of [BooleanExpressionType](#).

Key	Value
<code>name</code>	CustomTypeName
<code>objectType</code>	CustomTypeName
<code>dataConnectorName</code>	DataConnectorName
<code>dataConnectorObjectType</code>	DataConnectorObjectType

Key	Value
comparableFields	[ComparableField]
graphql	ObjectBooleanExpressionTypeGraphQLConfiguration / null

## ObjectBooleanExpressionTypeGraphQLConfiguration

GraphQL configuration of an Open DD boolean expression type.

Key	Value	Required	Description
typeName	GraphQLTypeName	true	The name to use for the GraphQL type representation of this boolean expression type.

## GraphQLTypeName

The name of a GraphQL type.

**Value:** string

## ComparableField

A field of an object type that can be used for comparison when evaluating a boolean expression.

Key	Value	Required	Description
fieldName	FieldName	true	
operators	EnableAllOrSpecific	true	Enable all or specific values.

## EnableAllOrSpecific

Enable all or specific values.

Must have exactly one of the following fields:

Key	Value	Required	Description
enableAll	boolean	false	
enableSpecific	[OperatorName]	false	

## OperatorName

The name of an operator

**Value:** string

## FieldName

The name of a field in a user-defined object type.

**Value:** string

## DataConnectorObjectType

The name of an object type in a data connector.

**Value:** string

## DataConnectorName

The name of a data connector.

**Value:** string

## **CustomTypeName**

The name of a user-defined type.

**Value:** string

# Aggregate Expressions

## Introduction

Aggregate expression types allow you to summarize and calculate collective properties of your data. These expressions enable you to define how to aggregate over different data types, facilitating efficient data manipulation and retrieval.

Aggregate expressions can be used to configure how data is summarized in your models, or as the types of arguments to commands or models.

## How AggregateExpressions work

### Lifecycle

#### EXAMPLES BELOW

Below this description of the lifecycle are examples for each step and how to define AggregateExpressions in your metadata files.

You'll need to ensure you've generated [ScalarTypes](#) and [DataConnectorScalarRepresentations](#) for the data types you want to aggregate over. We recommend saving these in your connector's type definition file.

Once you have these, you can create AggregateExpression objects for the types you want to aggregate over. We recommend these be saved in a model's metadata file.

Finally, you'll need to update your `graphql-config.hml` in the `globals` subgraph to include an `aggregate` field in the `definition`.

Once these are included, you'll need to [create a new build](#) using the CLI.

## Examples

In the following example, we'll show the snippets necessary to implement the example explained above: an AggregateExpression for the `Reviews` model of our sample application.

*metadata/my\_connector-types.html*

```
# Existing definitions above
---
kind: ScalarType
version: v1
definition:
  name: Numeric
  graphql:
    typeName: Numeric
---
kind: DataConnectorScalarRepresentation
version: v1
definition:
  dataConnectorName: my_connector
  dataConnectorScalarType: numeric
  representation: Numeric
  graphql:
    comparisonExpressionTypeName: NumericComparisonExp
```

*metadata/Reviews.html*

```
# Existing objects above
---
kind: AggregateExpression
version: v1
definition:
  name: Int4_aggregate_exp
  operand:
    scalar:
      aggregatedType: Int4
      aggregationFunctions:
        - name: avg
          returnType: Numeric
  dataConnectorAggregationFunctionMapping:
    - dataConnectorName: my_connector
      dataConnectorScalarType: int4
      functionMapping:
        avg:
          name: avg
```

```
graphql:
  selectTypeName: Int4_aggregate_fields
---
kind: AggregateExpression
version: v1
definition:
  name: Reviews_aggregate_exp
  operand:
    object:
      aggregatedType: Reviews
      aggregatableFields:
        - fieldName: rating
          aggregateExpression: Int4_aggregate_exp
graphql:
  selectTypeName: Reviews_aggregate_fields
description: Aggregate over Reviews
```

*globals/graphql-config.html*

```
kind: GraphqlConfig
version: v1
definition:
  query:
    rootOperationTypeName: Query
    argumentsInput:
      fieldName: args
    limitInput:
      fieldName: limit
    offsetInput:
      fieldName: offset
    filterInput:
      fieldName: where
    operatorNames:
      and: _and
      or: _or
      not: _not
      isNull: _is_null
    orderByInput:
      fieldName: order_by
    enumDirectionValues:
```

```

asc: Asc
desc: Desc
enumTypeNames:
  - directions:
    - Asc
    - Desc
  typeName: OrderBy
aggregate:
  filterInputFieldName: filter_input
  countFieldName: _count
  countDistinctFieldName: _count_distinct
mutation:
  rootOperationTypeName: Mutation

```

## Metadata structure

### AggregateExpression

Definition of an aggregate expression on an OpenDD type.

Key	Value	Required	Description
kind	AggregateExpression	true	
version	v1	true	
definition	AggregateExpressionV1	true	Definition of how to aggregate over a particular operand type

Example:

```

kind: AggregateExpression
version: v1
definition:
  name: Invoice_aggregate_exp

```

```

operand:
  object:
    aggregatedType: Invoice
    aggregatableFields:
      - fieldName: Total
        aggregateExpression: Float_aggregate_exp
graphql:
  selectTypeName: Invoice_aggregate_fields
description: Aggregate over Invoices

```

## AggregateExpressionV1

Definition of how to aggregate over a particular operand type

Key	Value	Required	Description
<code>name</code>	AggregateExpressionName	true	The name of the aggregate expression.
<code>operand</code>	AggregateOperand	true	The operand type being aggregated over.
<code>count</code>	AggregateCountDefinition / null	false	Count definition for the aggregate.
<code>countDistinct</code>	AggregateCountDefinition / null	false	Count distinct definition for the aggregate.
<code>graphql</code>	AggregateExpressionGraphQLDefinition / null	false	GraphQL definition for the aggregate.

Key	Value	Required	Description
			contains schema annotations
<code>description</code>	string / null	false	The description of an aggregate expression. Gets converted to the description of the corresponding scalar type.

## AggregateExpressionGraphQLDefinition

The definition of how an aggregate expression should appear in the GraphQL API.

Key	Value	Required	Description
<code>selectTypeName</code>	<a href="#">GraphQITypeName</a>	true	The type name to use for the aggregate selection type
<code>deprecated</code>	<a href="#">Deprecated</a> / null	false	Whether this command root field is deprecated. If set, this will be added to the graphql schema as a deprecated field.

**Example:**

```
selectTypeName: Invoice_aggregate_fields
```

## Deprecated

OpenDd configuration to indicate whether an object type field, relationship, model root field or command root field is deprecated.

Key	Value	Required	Description
reason	string / null	false	The reason for deprecation.

## GraphQLTypeName

The name of a GraphQL type.

Value: string

## AggregateCountDefinition

Definition of a count aggregation function

Key	Value	Required	Description
enable	boolean	true	Whether or not the aggregate function is available for use or not
description	string / null	false	A description of the aggregation function. Gets added to the description of the field in the GraphQL schema.

## AggregateOperand

Definition of an aggregate expression's operand

Must have exactly one of the following fields:

Key	Value	Required	Description
object	ObjectAggregateOperand	false	Definition of an aggregate over an object-typed operand

Key	Value	Required	Description
scalar	ScalarAggregateOperand	false	Definition of an aggregate over a scalar-typed operand

## ScalarAggregateOperand

Definition of an aggregate over a scalar-typed operand

Key	Value
aggregatedType	TypeName
aggregationFunctions	[AggregationFunctionDefinition]
dataConnectorAggregationFunctionMapping	[DataConnectorAggregationFunctionMapping]

## DataConnectorAggregationFunctionMapping

Definition of how to map an aggregate expression's aggregation functions to data connector aggregation functions.

Key	Value	Required
dataConnectorName	DataConnectorName	true

Key	Value	Required
<code>dataConnectorScalarType</code>	<code>DataConnectorScalarType</code>	true
<code>functionMapping</code>	<code>AggregationFunctionMappings</code>	true

## AggregationFunctionMappings

Mapping of aggregation functions to their matching aggregation functions in the data connector.

Key	Value	Required	Description
<code>&lt;customKey&gt;</code>	<code>AggregateFunctionMapping</code>	false	Definition of how to map the aggregation function to a function in the data connector

## AggregateFunctionMapping

Definition of how to map the aggregation function to a function in the data connector

Key	Value	Required	Description
name	DataConnectorAggregationFunctionName	true	The name of the aggregation function in the data connector

## DataConnectorAggregationFunctionName

The name of an aggregation function in a data connector

**Value:** string

## DataConnectorScalarType

The name of a scalar type in a data connector.

**Value:** string

## DataConnectorName

The name of a data connector.

**Value:** string

## AggregationFunctionDefinition

Definition of an aggregation function

Key	Value	Required	Description
name	AggregationFunctionName	true	The name of the aggregation function
description	string / null	false	A description of the aggregation function. Gets added to the description

Key	Value	Required	Description
			of the field in the GraphQL schema.
returnType	TypeReference	true	

## TypeReference

A reference to an Open DD type including nullable values and arrays. Suffix '!' to indicate a non-nullable reference, and wrap in '[]' to indicate an array. Eg: '[String!]!' is a non-nullable array of non-nullable strings.

**Value:** string

## AggregationFunctionName

The name of an aggregation function.

**Value:** string

## TypeName

The name of the scalar type the aggregate expression is aggregating

One of the following values:

Value	Description
InbuiltType	An inbuilt primitive OpenDD type.
CustomTypeName	

## InbuiltType

An inbuilt primitive OpenDD type.

**Value:** [ID](#) / [Int](#) / [Float](#) / [Boolean](#) / [String](#)

## ObjectAggregateOperand

Definition of an aggregate over an object-typed operand

Key	Value	Required	Description
aggregatedType	CustomTypeName	true	The name of the object type that is aggregated.
aggregatableFields	[AggregatableFieldDefinition]	true	The fields of the object type that are aggregated.

## AggregatableFieldDefinition

Definition of an aggregatable field on an object type

Key	Value	Required	Description
fieldName	FieldName	true	The name of the field on the object type that is aggregated.
description	string / null	false	A description of the aggregatable field. Can be added to describe the field in the GraphQL schema.
aggregateExpression	AggregateExpressionName	true	The aggregate expression used to aggregate the type of the field.

## **FieldName**

The name of a field in a user-defined object type.

**Value:** string

## **CustomTypeName**

The name of a user-defined type.

**Value:** string

## **AggregateExpressionName**

The name of an aggregate expression.

**Value:** string

# Relationships

## Introduction

Defining a relationship allows you to make queries across linked information.

Relationships are defined in metadata **from an object type, to a model or command**.

This enables you to create complex queries, where you can fetch related data in a single query.

By defining a `Relationship` object, all **models or commands** whose output type is the source object type defined in the relationship will now have a connection to the target model or command.

So from this relationship definition, all models or commands in the supergraph whose output type is the `orders` object type will now have a connection to the `customers` model via the `customer` relationship field on the `orders` type.

So, as in example 1 below, you can now fetch the customer when you query orders. Note that also, if you had, for instance an arbitrary command such as `getCustomerLastOrder` which also returned an order from a customer id, you can now also return the customer's details for that returned order in the same single query from this relationship.

## How relationships work

### Lifecycle

You can automatically add foreign-key relationships to your metadata [using the CLI](#).

Additionally, you can manually define relationships from an object type to a model or command in your metadata using the VS Code extension. The extension knows about your data sources and can help you write relationships more efficiently.

Simply start by typing a delimiter (`---`) at the end of a model's metadata file and start typing `Relationship`. The extension will provide you with auto-completion, syntax highlighting, and error checking as you write your relationships. At any point in the process, you can type

**Ctrl + Space** to see the available options and tab through the fields to fill in the required information.

To make a relationship available in your supergraph, you'll need to [create a new build](#) using the CLI.

## Examples

```
---
kind: Relationship
version: v1
definition:
  name: customer
  sourceType: orders
  target:
    model:
      name: customers
      subgraph: customers
      relationshipType: Object
  mapping:
    - source:
        fieldPath:
          - fieldName: customerId
    target:
      modelField:
        - fieldName: customerId
```

Field	Description	Reference
<b>kind</b>	Indicates that this object is defining a relationship between two data types or models.	<a href="#">Relationship</a>
<b>version</b>	Specifies the version of the	<a href="#">RelationshipV1</a>

Field	Description	Reference
	relationship's structure.	
<b>definition.name</b>	The name of the relationship, representing the link between the source and target.	<a href="#">RelationshipName</a>
<b>definition.sourceType</b>	Defines the source object type that the relationship starts from.	<a href="#">CustomTypeName</a>
<b>definition.target.model.name</b>	The name of the target model to which the source is related.	<a href="#">ModelName</a>
<b>definition.target.model.subgraph</b>	Specifies the subgraph in which the target model resides.	<a href="#">ModelRelationshipTail</a>
<b>definition.target.model.relationshipType</b>	Indicates the type of relationship (Object or Array), determining whether one or many related items can be fetched.	<a href="#">RelationshipType</a>
<b>definition.mapping</b>	Defines how fields from the source map to fields or arguments	<a href="#">RelationshipMapping</a>

Field	Description	Reference
	in the target, establishing the connection between them.	

## Object Type to a Model

As mentioned above, in an e-commerce context, you will likely have customers and orders, and you want to relate them so that when you query orders you can easily fetch the customer on that order.

To do this in DDN metadata for this example, you might have an `orders` model which returns an `orders` object type and you want to relate that to a `customers` model and whatever object type it returns.

**Example:** Fetch a list of orders along with the customer details for each order:

Here is the corresponding relationship configuration which enables this query:

```

kind: Relationship
version: v1
definition:
  name: customer
  sourceType: orders
  target:
    model:
      name: customers
      subgraph: customers
      relationshipType: Object
  mapping:
    - source:
        fieldPath:
          - fieldName: customerId
    target:
      modelField:
        - fieldName: customerId
description: The customer details for an order

```

Now, you can also retrieve the customer details for any model or command in your schema that returns the `orders` object type.

For convenience, this relationship configuration would best be located with your `orders` object type and model.

## Object Type to a Command

Let's say you have a `user` object type in your graph, and also a command which responds with the current logged-in user information (say `getLoggedInUserInfo`). Now you can link these two, by defining a relationship from the `user` object type to `getLoggedInUserInfo`.

Let's say we name the relationship `currentSession`. Now you can make a single query from your client to get a user's data and their current session information.

**Example:** fetch a list of users and the current session information of each user:

Here is the corresponding relationship configuration which enables this query:

```
kind: Relationship
version: v1
definition:
  name: currentSession
  sourceType: user
  target:
    command:
      name: getLoggedInUserInfo
      subgraph: users
  mapping:
    - source:
        fieldPath:
          - fieldName: id
    target:
      argument:
        argumentName: user_id
description: The current session information for the user
```

## Command to Command

Hasura DDN also allows you to link commands together from a source type to query related data.

**Example:** fetch the result of one command and use it as input for another command:

And the corresponding relationship configuration which enables this query:

```
kind: Relationship
version: v1
definition:
  name: shippingDetails
  sourceType: trackOrder
  target:
    command:
      name: getShippingDetails
      subgraph: orders
  mapping:
    - source:
        fieldPath:
          - fieldName: trackingNumber
    target:
      argument:
        argumentName: trackingNumber
  description: The shipping details for an order based on its tracking number
```

### (!) VS CODE EXTENSION ASSISTED AUTHORING

Relationships are written in your various HML files. If you're using a compatible editor (such as VS Code with the [Hasura VS Code extension](#)), assisted authoring will help you create relationships more efficiently. It will provide you with auto-completion, syntax highlighting, and error checking as you write your relationships.

The CLI also works to automatically track your relationships for you whenever you add or update a data connector.

# Metadata structure

## Relationship

Definition of a relationship on an OpenDD type which allows it to be extended with related models or commands.

Key	Value	Required	Description
kind	Relationship	true	
version	v1	true	
definition	RelationshipV1	true	Definition of a relationship on an OpenDD type which allows it to be extended with related models or commands.

Example:

```
kind: Relationship
version: v1
definition:
  name: Articles
  sourceType: author
  target:
    model:
      name: Articles
      subgraph: null
      relationshipType: Array
  mapping:
    - source:
        fieldPath:
          - fieldName: author_id
    target:
      modelField:
        - fieldName: author_id
description: Articles written by an author
```

## RelationshipV1

Definition of a relationship on an OpenDD type which allows it to be extended with related models or commands.

Key	Value	Required	Description
name	RelationshipName	true	The name of the relationship.
sourceType	CustomTypeName	true	The source type of the relationship.
target	RelationshipTarget	true	The target of the relationship.
mapping	[RelationshipMapping]	true	The mapping configuration of source to target for the relationship.
description	string / null	false	The description of the relationship. Gets added to the description of the relationship in the graphql schema.
deprecated	Deprecated / null	false	Whether this relationship is deprecated. If set, the deprecation status is added to the relationship field's

Key	Value	Required	Description
			graphql schema.
graphql	RelationshipGraphQLDefinition / null	false	Configuration for how this relationship should appear in the GraphQL schema.

## RelationshipGraphQLDefinition

The definition of how a relationship appears in the GraphQL API

Key	Value	Required	Description
aggregateFieldName	FieldName / null	false	The field name to use for the field that represents an aggregate over the relationship

## Deprecated

OpenDd configuration to indicate whether an object type field, relationship, model root field or command root field is deprecated.

Key	Value	Required	Description
reason	string / null	false	The reason for deprecation.

## RelationshipMapping

Definition of a how a particular field in the source maps to a target field or argument.

Key	Value	Required	Description
source	RelationshipMappingSource	true	The source configuration for this relationship mapping.
target	RelationshipMappingTarget	true	The target configuration for this relationship mapping.

Example:

```
source:
  fieldPath:
    - fieldName: author_id
target:
  modelField:
    - fieldName: author_id
```

## RelationshipMappingTarget

The target configuration for a relationship mapping.

Must have exactly one of the following fields:

Key	Value	Required	Description
argument	ArgumentMappingTarget	false	An argument target for a relationship mapping.
modelField	[RelationshipSourceFieldAccess]	false	

## ArgumentMappingTarget

An argument target for a relationship mapping.

Key	Value	Required	Description
argumentName	ArgumentName	true	

## ArgumentName

The name of an argument.

**Value:** string

## RelationshipMappingSource

The source configuration for a relationship mapping.

Must have exactly one of the following fields:

Key	Value	Required	Description
value	ValueExpression	false	An expression which evaluates to a value that can be used in permissions and various presets.
fieldPath	[RelationshipSourceFieldAccess]	false	

## RelationshipSourceFieldAccess

A field access in a relationship mapping.

Key	Value	Required	Description
fieldName	FieldName	true	

## FieldName

The name of a field in a user-defined object type.

**Value:** string

## ValueExpression

An expression which evaluates to a value that can be used in permissions and various presets.

Must have exactly one of the following fields:

Key	Value	Required	Description
literal		false	
sessionVariable	string	false	

## RelationshipTarget

The target for a relationship.

Must have exactly one of the following fields:

Key	Value	Required	Description
model	ModelRelationshipTarget	false	The target model for a relationship.
command	CommandRelationshipTarget	false	The target command for a relationship.

Example:

```
model:  
  name: Articles  
  subgraph: null  
  relationshipType: Array
```

## CommandRelationshipTarget

The target command for a relationship.

Key	Value	Required	Description
name	CommandName	true	The name of the command.
subgraph	string / null	false	The subgraph of the target command. Defaults to the current subgraph.

## CommandName

The name of a command.

**Value:** string

## ModelRelationshipTarget

The target model for a relationship.

Key	Value	Required	Description
name	ModelName	true	The name of the target model.
subgraph	string / null	false	The subgraph of the target model. Defaults to the current subgraph.
relationshipType	RelationshipType	true	Type of relationship - can be an array.
aggregate	ModelRelationshipTargetAggregate / null	false	How aggregate over relationships. Only for relationships.

## ModelRelationshipTargetAggregate

Which aggregate expression to use to aggregate the array relationship.

Key	Value	Required	Description
aggregateExpression	AggregateExpressionName	true	The name of the aggregate expression that defines how to aggregate across relationships.
description	string / null	false	The description of the relationship aggregation. Gets a reference to the description of the aggregate field in the Graph schema.

## AggregateExpressionName

The name of an aggregate expression.

**Value:** string

## RelationshipType

Type of the relationship.

**One of the following values:**

Value	Description
Object	Select one related object from the target.
Array	Select multiple related objects from the target.

## **modelName**

The name of data model.

**Value:** string

## **customTypeName**

The name of a user-defined type.

**Value:** string

## **relationshipName**

The name of the GraphQL relationship field.

**Value:** string

# Permissions

## Introduction

Access control rules are essential for securing your data and ensuring that only authorized users can access it. In Hasura, these are referred to as **permissions**. You can use permissions to control access to certain rows or columns in your database, or to restrict access to certain operations or fields in your GraphQL API.

The following types of permissions can be defined:

Type	Description
TypePermissions	Define which fields are allowed to be accessed by a role on an output type.
ModelPermissions	Define which objects or rows within a model are allowed to be accessed by a role.
CommandPermissions	Define whether the command is executable by a role.

## How TypePermissions work

### Lifecycle

By default, whenever a new type is created in your supergraph, each field is only accessible to the `admin` role. You can think of these as permissions on columns in a database table.

You can quickly generate new roles by adding a new item to the `permissions` array in the `TypePermissions` object. Each item in the array should have a `role` field and an `output` field. The `output` field should contain an `allowedFields` array, which lists the fields that are accessible to the role when the type is used in an output context.

To make a new TypePermission or role available in your supergraph, you'll need to [create a new build](#) using the CLI.

### Examples

```
---
kind: TypePermissions
version: v1
definition:
  typeName: article
  permissions:
    - role: admin
      output:
        allowedFields:
          - article_id
          - author_id
          - title
    - role: user
      output:
        allowedFields:
          - article_id
          - author_id
```

Field	Description	Reference
<b>kind</b>	Indicates that this object is defining permissions for a specific type.	TypePermissions
<b>version</b>	Specifies the version of the type permissions structure.	TypePermissions
<b>definition.typeName</b>	The name of the type for which permissions are being defined.	CustomTypeName
<b>definition.permissions</b>	A list of permissions	TypePermission

Field	Description	Reference
	specifying which fields are accessible for different roles.	
<code>definition.permissions[].role</code>	The role for which permissions are being defined.	<a href="#">Role</a>
<code>definition.permissions[].output.allowedFields</code>	The fields of the type that are accessible for a role when the type is used in an output context.	<a href="#">TypeOutputPerm</a>

## How ModelPermissions work

### Lifecycle

By default, whenever a new model is created in your supergraph, all records are only accessible to the `admin` role. You can think of these as permissions on rows in a database table.

You can restrict access to certain rows by adding a new item to the `permissions` array in the `ModelPermissions` object. Each item in the array should have a `role` field and a `select` field. The `select` field should contain a `filter` expression that determines which rows are accessible to the role when selecting from the model.

Most commonly, you'll use session variables — passed in the header of a request — to restrict access to rows based on the user's identity or other criteria. You can also use argument presets to enforce row-level security. You can see an example of this below.

To make a new ModelPermission or role available in your supergraph, after updating your metadata, you'll need to [create a new build](#) using the CLI.

## Examples

```
---  
kind: ModelPermissions  
version: v1  
definition:  
  modelName: Articles  
  permissions:  
    - role: admin  
      select:  
        filter: null  
    - role: user  
      select:  
        filter:  
          fieldComparison:  
            field: author_id  
            operator: _eq  
            value:  
              sessionVariable: x-hasura-user-id
```

Field	Description	Reference
kind	Indicates that this object is defining permissions for a specific model.	<a href="#">ModelPermissions</a>
version	Specifies the version of the model permissions structure.	<a href="#">ModelPermissionsV1</a>
definition.modelName	The name of the model for which permissions	<a href="#">modelName</a>

Field	Description	Reference
	are being defined.	
<code>definition.permissions</code>	A list of permissions specifying which rows or objects are accessible for different roles.	<a href="#">ModelPermission</a>
<code>definition.permissions[].role</code>	The role for which permissions are being defined.	<a href="#">Role</a>
<code>definition.permissions[].select.filter</code>	A filter expression that determines which rows are accessible for this role when selecting from the model.	<a href="#">ModelPredicate</a>

## How CommandPermissions work

### Lifecycle

By default, whenever a new command is created in your supergraph, it is only executable by the `admin` role.

You can enable or restrict access to commands by adding a new item to the `permissions` array in the `CommandPermissions` object. Each item in the array should have a `role` field and an `allowExecution` field. The `allowExecution` field should be set to `true` if the command is executable by the role.

Further, you can use argument presets to pass actual logical expressions to your data sources to control how they do things.

For example, a data connector might expose a `Command` called `delete_user_by_id` with two arguments - `user_id` and `pre_check`. `user_id` is the primary key of the user you'd like to remove, and `pre_check` lets you provide a custom boolean expression.

```
kind: CommandPermissions
version: v1
definition:
  commandName: delete_user_by_id
  permissions:
    - role: admin
      allowExecution: true
    - role: user
      allowExecution: true
  argumentPresets:
    - argument: pre_check
      value:
        booleanExpression:
          fieldComparison:
            field: is_invincible
            operator: _eq
            value:
              literal: false
```

Now, when `admin` runs this command, once again, they can do what they want, and provide their own `pre_check` if they want. The `user` however, is able to pass a `user_id` argument, but the `pre_check` expression is passed to the data connector which will only let them delete the row if the row's `is_invincible` value is set to `false`.

To make a execution of a command available to a role in your supergraph, after updating your metadata, you'll need to [create a new build](#) using the CLI.

## Examples

```
---
kind: CommandPermissions
version: v1
definition:
  commandName: get_article_by_id
```

```

permissions:
  - role: admin
    allowExecution: true
  - role: user
    allowExecution: true
  argumentPresets:
    - argument: id
      value:
        literal: 100

```

Field	Description	Reference
<b>kind</b>	Indicates that this object is defining permissions for a specific command.	<a href="#">CommandPermission</a>
<b>version</b>	Specifies the version of the command permissions structure.	<a href="#">CommandPermission</a>
<b>definition.commandName</b>	The name of the command for which permissions are being defined.	<a href="#">CommandName</a>
<b>definition.permissions</b>	A list of permissions specifying whether the command is executable by different roles.	<a href="#">CommandPermission</a>

Field	Description	Reference
<code>definition.permissions[].role</code>	The role for which permissions are being defined.	<a href="#">Role</a>
<code>definition.permissions[].allowExecution</code>	Indicates whether the command is executable by the role.	<a href="#">CommandPermission</a>
<code>definition.permissions[].argumentPresets</code>	Preset values for arguments that are enforced when the command is executed by the role.	<a href="#">ArgumentPreset</a>

## Metadata structure

### TypePermissions

Definition of permissions for an OpenDD type.

Key	Value	Required	Description
<code>kind</code>	<code>TypePermissions</code>	true	
<code>version</code>	<code>v1</code>	true	
<code>definition</code>	<code>TypePermissionsV1</code>	true	Definition of permissions for an OpenDD type.

Example:

```

kind: TypePermissions
version: v1
definition:
  typeName: article
  permissions:
    - role: admin
      output:
        allowedFields:
          - article_id
          - author_id
          - title
    - role: user
      output:
        allowedFields:
          - article_id
          - author_id

```

## TypePermissionsV1

Definition of permissions for an OpenDD type.

Key	Value	Required	Description
<code>typeName</code>	CustomTypeName	true	The name of the type for which permissions are being defined. Must be an object type.
<code>permissions</code>	[TypePermission]	true	A list of type permissions, one for each role.

## TypePermission

Defines permissions for a particular role for a type.

Key	Value	Required	Description
role	Role	true	The role for which permissions are being defined.
output	TypeOutputPermission / null	false	Permissions for this role when this type is used in an output context. If null, this type is inaccessible for this role in an output context.
input	TypeInputPermission / null	false	Permissions for this role when this type is used in an input context. If null, this type is accessible for this role in an input context.

Example:

```
role: user
output:
  allowedFields:
    - article_id
    - author_id
input:
  fieldPresets:
    - field: author_id
      value:
        sessionVariable: x-hasura-user-id
```

## TypeInputPermission

Permissions for a type for a particular role when used in an input context.

Key	Value	Required	Description
fieldPresets	[FieldPreset]	false	Preset values for fields of the type

## FieldPreset

Preset value for a field

Key	Value	Required	Description
field	FieldName	true	Field name for preset
value	ValueExpression	true	Value for preset

## ValueExpression

An expression which evaluates to a value that can be used in permissions and various presets.

Must have exactly one of the following fields:

Key	Value	Required	Description
literal		false	
sessionVariable	string	false	

## TypeOutputPermission

Permissions for a type for a particular role when used in an output context.

Key	Value	Required	Description
allowedFields	[FieldName]	true	Fields of the type that are accessible for a role

## FieldName

The name of a field in a user-defined object type.

**Value:** string

## Role

The role for which permissions are being defined.

**Value:** string

## CustomTypeName

The name of a user-defined type.

**Value:** string

## ModelPermissions

Definition of permissions for an OpenDD model.

Key	Value	Required	Description
kind	ModelPermissions	true	
version	v1	true	
definition	ModelPermissionsV1	true	Definition of permissions for an OpenDD model.

**Examples:**

```
kind: ModelPermissions
version: v1
definition:
  modelName: Articles
  permissions:
    - role: admin
      select:
        filter: null
    - role: user
      select:
        filter:
          fieldComparison:
            field: author_id
            operator: _eq
            value:
              sessionVariable: x-hasura-user-id
```

```

kind: ModelPermissions
version: v1
definition:
  modelName: Articles
  permissions:
    - role: admin
      select:
        filter: null
    - role: user
      select:
        filter:
          relationship:
            name: author
            predicate:
              fieldComparison:
                field: id
                operator: _eq
                value:
                  sessionVariable: x-hasura-user-id

```

## ModelPermissionsV1

Definition of permissions for an OpenDD model.

Key	Value	Required	Description
<code>modelName</code>	<code>ModelName</code>	true	The name of the model for which permissions are being defined.
<code>permissions</code>	<code>[ModelPermission]</code>	true	A list of model permissions, one for each role.

## ModelPermission

Defines the permissions for an OpenDD model.

Key	Value	Required	Description
role	Role	true	The role for which permissions are being defined.
select	SelectPermission / null	false	The permissions for selecting from this model for this role. If this is null, the role is not allowed to query the model.

Example:

```
role: user
select:
  filter:
    fieldComparison:
      field: author_id
      operator: _eq
      value:
        sessionVariable: x-hasura-user-id
argument_presets:
  - field: likes_dogs
    value:
      literal: true
```

## SelectPermission

Defines the permissions for selecting a model for a role.

Key	Value	Required	Description
filter	null / ModelPredicate	true	Filter expression when selecting rows for this model. Null filter implies all rows are selectable.

Key	Value	Required	Description
argumentPresets	[ArgumentPreset]	false	Preset values for arguments for this role
allowSubscriptions	boolean	false	Whether the role is allowed to subscribe to the root fields of this model.

## ArgumentPreset

Preset value for an argument

Key	Value	Required	Description
argument	ArgumentName	true	Argument name for preset
value	ValueExpressionOrPredicate	true	Value for preset

## ValueExpressionOrPredicate

An expression which evaluates to a value that can be used in permissions and various presets.

Must have exactly one of the following fields:

Key	Value	Required	Description
literal		false	
sessionVariable	string	false	Used to represent the name of a session variable, like "x-hasura-role".
booleanExpression	ModelPredicate	false	

## ArgumentName

The name of an argument.

**Value:** string

## ModelPredicate

A predicate that can be used to restrict the objects returned when querying a model.

Must have exactly one of the following fields:

Key	Value	Required	Description
fieldComparison	FieldComparisonPredicate	false	Field comparison predicate filters objects based on a field value.
fieldIsNull	FieldIsNullPredicate	false	Predicate to check if the given field is null.
relationship	RelationshipPredicate	false	Relationship predicate filters objects of a source model based on a predicate on the related model.
and	[ModelPredicate]	false	
or	[ModelPredicate]	false	
not	ModelPredicate	false	

Examples:

```
fieldComparison:  
  field: author_id  
  operator: _eq  
  value:  
    sessionVariable: x-hasura-user-id
```

```
relationship:  
  name: author  
  predicate:  
    fieldComparison:  
      field: id  
      operator: _eq  
      value:  
        sessionVariable: x-hasura-user-id
```

```
and:  
- fieldComparison:  
  field: author_id  
  operator: _eq  
  value:  
    sessionVariable: x-hasura-user-id  
- fieldComparison:  
  field: title  
  operator: _eq  
  value:  
    literal: Hello World
```

```
not:  
  fieldComparison:  
    field: author_id  
    operator: _eq  
    value:  
      sessionVariable: x-hasura-user-id
```

## RelationshipPredicate

Relationship predicate filters objects of a source model based on a predicate on the related model.

Key	Value	Required	Description
name	RelationshipName	true	The name of the relationship of the object type of the model to follow.
predicate	ModelPredicate / null	false	The predicate to apply on the related objects. If this is null, then the predicate evaluates to true as long as there is at least one related object present.

## RelationshipName

The name of the GraphQL relationship field.

**Value:** string

## FieldIsNullPredicate

Predicate to check if the given field is null.

Key	Value	Required	Description
field	FieldName	true	The name of the field that should be checked for a null value.

## FieldComparisonPredicate

Field comparison predicate filters objects based on a field value.

Key	Value	Required	Description
field	FieldName	true	The field name of the object type of the model to compare.

Key	Value	Required	Description
operator	OperatorName	true	The name of the operator to use for comparison.
value	ValueExpression	true	The value expression to compare against.

## ValueExpression

An expression which evaluates to a value that can be used in permissions and various presets.

Must have exactly one of the following fields:

Key	Value	Required	Description
literal		false	
sessionVariable	string	false	

## OperatorName

The name of an operator

**Value:** string

## FieldName

The name of a field in a user-defined object type.

**Value:** string

## Role

The role for which permissions are being defined.

**Value:** string

## modelName

The name of data model.

**Value:** string

## CommandPermissions

Definition of permissions for an OpenDD command.

Key	Value	Required	Description
kind	CommandPermissions	true	
version	v1	true	
definition	CommandPermissionsV1	true	Definition of permissions for an OpenDD command.

**Example:**

```
kind: CommandPermissions
version: v1
definition:
  commandName: get_article_by_id
  permissions:
    - role: admin
      allowExecution: true
    - role: user
      allowExecution: true
```

## CommandPermissionsV1

Definition of permissions for an OpenDD command.

Key	Value	Required	Description
commandName	CommandName	true	The name of the command for which permissions are being defined.

Key	Value	Required	Description
permissions	[CommandPermission]	true	A list of command permissions, one for each role.

## CommandPermission

Defines the permissions for a role for a command.

Key	Value	Required	Description
role	Role	true	The role for which permissions are being defined.
allowExecution	boolean	true	Whether the command is executable by the role.
argumentPresets	[ArgumentPreset]	false	Preset values for arguments for this role

Example:

```
role: user
allowExecution: true
argumentPresets:
  - argument: user_id
    value:
      session_variable: x-hasura-user_id
```

## ArgumentPreset

Preset value for an argument

Key	Value	Required	Description
argument	ArgumentName	true	Argument name for preset
value	ValueExpressionOrPredicate	true	Value for preset

## ValueExpressionOrPredicate

An expression which evaluates to a value that can be used in permissions and various presets.

Must have exactly one of the following fields:

Key	Value	Required	Description
literal		false	
sessionVariable	string	false	Used to represent the name of a session variable, like "x-hasura-role".
booleanExpression	ModelPredicate	false	

## ModelPredicate

A predicate that can be used to restrict the objects returned when querying a model.

Must have exactly one of the following fields:

Key	Value	Required	Description
fieldComparison	FieldComparisonPredicate	false	Field comparison predicate filters objects based on a field value.
fieldIsNull	FieldIsNullPredicate	false	Predicate to check if the

Key	Value	Required	Description
			given field is null.
relationship	RelationshipPredicate	false	Relationship predicate filters objects of a source model based on a predicate on the related model.
and	[ModelPredicate]	false	
or	[ModelPredicate]	false	
not	ModelPredicate	false	

### Examples:

```
fieldComparison:
  field: author_id
  operator: _eq
  value:
    sessionVariable: x-hasura-user-id
```

```
relationship:
  name: author
  predicate:
    fieldComparison:
      field: id
      operator: _eq
      value:
        sessionVariable: x-hasura-user-id
```

and:

```
- fieldComparison:  
  field: author_id  
  operator: _eq  
  value:  
    sessionVariable: x-hasura-user-id  
- fieldComparison:  
  field: title  
  operator: _eq  
  value:  
    literal: Hello World
```

not:

```
fieldComparison:  
  field: author_id  
  operator: _eq  
  value:  
    sessionVariable: x-hasura-user-id
```

## RelationshipPredicate

Relationship predicate filters objects of a source model based on a predicate on the related model.

Key	Value	Required	Description
name	RelationshipName	true	The name of the relationship of the object type of the model to follow.
predicate	ModelPredicate / null	false	The predicate to apply on the related objects. If this is null, then the predicate evaluates to true as long as there is at least one related object present.

## RelationshipName

The name of the GraphQL relationship field.

**Value:** string

## FieldIsNotNullPredicate

Predicate to check if the given field is null.

Key	Value	Required	Description
field	FieldName	true	The name of the field that should be checked for a null value.

## FieldComparisonPredicate

Field comparison predicate filters objects based on a field value.

Key	Value	Required	Description
field	FieldName	true	The field name of the object type of the model to compare.
operator	OperatorName	true	The name of the operator to use for comparison.
value	ValueExpression	true	The value expression to compare against.

## ValueExpression

An expression which evaluates to a value that can be used in permissions and various presets.

Must have exactly one of the following fields:

Key	Value	Required	Description
literal		false	
sessionVariable	string	false	

## OperatorName

The name of an operator

**Value:** string

## **FieldName**

The name of a field in a user-defined object type.

**Value:** string

## **ArgumentName**

The name of an argument.

**Value:** string

## **Role**

The role for which permissions are being defined.

**Value:** string

## **CommandName**

The name of a command.

**Value:** string

# Configuration for GraphQL API

## Introduction

The `GraphqlConfig` object allows you to configure the GraphQL schema generated by Hasura. This object helps you configure the common GraphQL schema applicable to all subgraphs within a project.

The `GraphqlConfig` object belongs to the supergraph and can be defined **once** across your supergraph in any subgraph of your choice.

## How GraphqlConfig works

### Lifecycle

By default, all projects are created with a default `GraphqlConfig` object in the `globals` subgraph.

Specifically, it allows you to configure:

1. The root type names for query, mutation and subscription operations.
2. The name for `model` selection input field names (`where`, `limit`, `offset`, `args`, `order_by`)
3. The enum values for `order_by` directions and the `order_by` enum type name.

An `GraphqlConfig` object is required to be defined in the supergraph metadata. If not defined, any attempted builds will not be successful.

## Examples

```
kind: GraphqlConfig
version: v1
definition:
  query:
    rootOperationTypeName: Query
    argumentsInput:
      fieldName: args
    limitInput:
      fieldName: limit
```

```

offsetInput:
  fieldName: offset
filterInput:
  fieldName: where
operatorNames:
  and: _and
  or: _or
  not: _not
  isNull: _is_null
orderByInput:
  fieldName: order_by
enumDirectionValues:
  asc: Asc
  desc: Desc
enumTypeNames:
  - directions:
    - Asc
    - Desc
    typeName: OrderBy
mutation:
  rootOperationTypeName: Mutation
apolloFederation:
  enableRootFields: false

```

Field	Description
kind	The type of configuration you're defining, in this case, it's for GraphQL API settings.
version	The version of this configuration object.
definition.query.rootOperationTypeName	The name used for the root query type in your GraphQL

Field	Description
	schema, usually <code>Query</code> .
<code>definition.query.argumentsInput.fieldName</code>	The field name used for passing arguments in queries, usually <code>args</code> .
<code>definition.query.limitInput.fieldName</code>	The field name used to limit the number of results, typically <code>limit</code> .
<code>definition.query.offsetInput.fieldName</code>	The field name used to set the starting point for results, typically <code>offset</code> .
<code>definition.query.filterInput.fieldName</code>	The field name used for filtering results, usually <code>where</code> .
<code>definition.query.filterInput.operatorNames []</code>	The names of the built-in filter operators.
<code>definition.query.orderByInput.fieldName</code>	The field name used for sorting results, usually <code>order_by</code> .

Field	Description
<code>definition.query.orderByInput.enumDirectionValues[]</code>	The names of the direction parameters, like <code>asc</code> and <code>desc</code> .
<code>definition.query.orderByInput.enumTypeNames</code>	The name of the enum type used for sorting, like <code>OrderBy</code> .
<code>definition.mutation.rootOperationTypeName</code>	The name used for the root mutation type in your GraphQL schema, usually <code>Mutation</code> .
<code>definition.apolloFederation.enableRootFields</code>	A setting to enable or disable fields needed for Apollo Federation ( <code>_entities</code> , <code>_services</code> ).

## Metadata structure

### GraphqlConfig

GraphqlConfig object tells us two things:

1. How the Graphql schema should look like for the features (`where`, `order_by` etc)  
Hasura provides
2. What features should be enabled/disabled across the subgraphs

Key	Value	Required	Description
kind	GraphqlConfig	true	
version	v1	true	
definition	GraphQLConfigV1	true	GraphQLConfig object tells us two things: 1. How the Graphql schema should look like for the features (where, order_by etc) Hasura provides 2. What features should be enabled/disabled across the subgraphs

## GraphQLConfigV1

GraphQLConfig object tells us two things:

1. How the Graphql schema should look like for the features (where, order\_by etc) Hasura provides
2. What features should be enabled/disabled across the subgraphs

Key	Value	Required	Description
query	QueryGraphQLConfig	true	Config for the Graphql schema. Hasura features queries. None means disable feature.
mutation	MutationGraphQLConfig	true	Config for the Graphql schema. Hasura features mutations.

Key	Value	Required	Description
subscription	SubscriptionGraphqlConfig / null	false	
apolloFederation	GraphqlApolloFederationConfig / null	false	

## GraphqlApolloFederationConfig

Configuration for the GraphQL schema of Hasura features for Apollo Federation.

Key	Value	Required	Description
enableRootFields	boolean	true	Adds the <code>_entities</code> and <code>_services</code> root fields required for Apollo Federation.

## SubscriptionGraphqlConfig

Configuration for the GraphQL schema of Hasura features for subscriptions.

Key	Value	Required	Description
rootOperationTypeName	GraphQLTypeName	true	The name of the root operation type name for subscription. Usually <code>subscribe</code> .

## MutationGraphqlConfig

Configuration for the GraphQL schema of Hasura features for mutations.

Key	Value	Required	Description
rootOperationTypeName	GraphQLTypeName	true	The name of the root operation type name for

Key	Value	Required	Description
			mutations. Usually <code>mutation</code>

## QueryGraphqlConfig

Configuration for the GraphQL schema of Hasura features for queries. `None` means disable the feature.

Key	Value	Required
<code>rootOperationTypeName</code>	<code>GraphQLType</code>	true
<code>argumentsInput</code>	<code>ArgumentsInputGraphqlConfig / null</code>	false
<code>limitInput</code>	<code>LimitInputGraphqlConfig / null</code>	false
<code>offsetInput</code>	<code>OffsetInputGraphqlConfig / null</code>	false
<code>filterInput</code>	<code>FilterInputGraphqlConfig / null</code>	false
<code>orderByInput</code>	<code>OrderByInputGraphqlConfig / null</code>	false
<code>aggregate</code>	<code>AggregateGraphqlConfig / null</code>	false

## AggregateGraphqlConfig

Configuration for the GraphQL schema for aggregates.

Key	Value	Required	Description
filterInputFieldName	GraphQLFieldName	true	The name of the filter input parameter for aggregate fields and name in predicates
countFieldName	GraphQLFieldName	true	The name of the _count field used in the count aggregate function
countDistinctFieldName	GraphQLFieldName	true	The name of the _count_distinct field used in the count distinct aggregate function

## OrderByInputGraphqlConfig

Configuration for the sort operation.

Key	Value	Required	Description
fieldName	GraphQLFieldName	true	The name of the field for the sort operation. Usually order by.
enumDirectionValues	OrderByDirectionValues	true	The name of the direction values parameter

Key	Value	Required	Description
enumTypeNames	[OrderByEnumTypeName]	true	

## OrderByEnumTypeName

Type name for a sort directions enum, with the given set of possible directions.

Key	Value	Required	Description
directions	[OrderByDirection]	true	
typeName	GraphQLTypeName	true	

## OrderByDirection

Sort direction.

One of the following values:

Value	Description
Asc	Ascending.
Desc	Descending.

## OrderByDirectionValues

The names of the direction parameters.

Key	Value	Required	Description
asc	GraphQLFieldName	true	The name of the ascending parameter. Usually Asc.
desc	GraphQLFieldName	true	The name of the descending parameter. Usually Desc.

## FilterInputGraphqlConfig

Configuration for the filter operation.

Key	Value	Required	Description
fieldName	GraphQLFieldName	true	The name of the filter operation field. Usually <code>where</code> .
operatorNames	FilterInputOperatorNames	true	The names of built-in filter operators.

## FilterInputOperatorNames

The names of built-in filter operators.

Key	Value	Required	Description
and	GraphQLFieldName	true	The name of the <code>and</code> operator. Usually <code>_and</code> .
or	GraphQLFieldName	true	The name of the <code>or</code> operator. Usually <code>_or</code> .
not	GraphQLFieldName	true	The name of the <code>not</code> operator. Usually <code>_not</code> .
isNull	GraphQLFieldName	true	The name of the <code>is null</code> operator. Usually <code>_is_null</code> .

## OffsetInputGraphqlConfig

Configuration for the offset operation.

Key	Value	Required	Description
fieldName	GraphQLFieldName	true	The name of the offset operation field. Usually <code>offset</code> .

## LimitInputGraphQLConfig

Configuration for the limit operation.

Key	Value	Required	Description
fieldName	GraphQLFieldName	true	The name of the limit operation field. Usually <code>limit</code> .

## ArgumentsInputGraphQLConfig

Configuration for the arguments input.

Key	Value	Required	Description
fieldName	GraphQLFieldName	true	The name of arguments passing field. Usually <code>args</code> .

## GraphQLFieldName

The name of a GraphQL object field.

**Value:** string

## GraphQLTypeName

The name of a GraphQL type.

**Value:** string

# AuthConfig

## Introduction

The `AuthConfig` object is used to define the authentication configuration used by the API. With Hasura, you can utilize either webhooks or JWTs for authentication. **The AuthConfig object belongs to the supergraph** and can be defined **once** across your supergraph in **any subgraph** of your choice using one of the following modes:

Mode	Description
<code>noAuth</code>	This is enabled by default and allows you to connect to your API without any authentication. <b>It is not recommended for production.</b>
<code>JWT</code>	This allows you to use JWTs for authentication.
<code>Webhook</code>	This allows you to use a custom webhook for authentication.

You can learn more about Hasura's approach to using existing authentication systems in the [authentication section](#).

## How AuthConfig works

### Lifecycle

By default, all projects are created with an `AuthConfig` object in the **globals** subgraph in a `noAuth` mode that doesn't use any auth service and doesn't allow setting session variables with API requests. Meaning that your API is, by default, fully public and exposed if deployed live.

However, you would want update the `AuthConfig` to use a custom webhook or JWT service for authentication to restrict access to your API and make use of Hasura's powerful [authorization](#) features. The metadata examples below can help you configure your `AuthConfig` object to use your own custom webhook or JWT service.

An `AuthConfig` object is required to be defined in the supergraph metadata. If not defined, any attempted builds will not be successful.

To make an update AuthConfig available in your supergraph, after updating your metadata, you'll need to [create a new build](#) using the CLI.

## Examples

Each example below offers numerous customizations. However, we've provided a basic example for each mode to get you started. Check the reference table for more information on each field.

### noAuth

*An AuthConfig for noAuth mode:*

```
kind: AuthConfig
version: v2
definition:
  mode:
    noAuth:
      role: admin
      sessionVariables: {}
```

Field	Description	Reference
<code>role</code>	The role to be assumed while running the engine in <code>noAuth</code> mode.	<a href="#">Role</a>
<code>sessionVariables</code>	Static session variables that will be used while running the engine without authentication. This is helpful when you want to test requests using particular session variables, such as <code>x-hasura-user-id</code> with a non-admin role.	<a href="#">SessionVariables</a>

### JWT

*An AuthConfig for JWT mode:*

```
kind: AuthConfig
version: v2
```

```

definition:
  mode:
    jwt:
      claimsConfig:
        namespace:
          claimsFormat: Json
          location: /claims.jwt.hasura.io
      tokenLocation:
        type: BearerAuthorization
      key:
        fixed:
          algorithm: HS256
        key:
          valueFromEnv: AUTH_SECRET

```

Field	Description	Reference
<b>claimsConfig</b>	Configuration to describe how and where the engine should look for the claims within the decoded token. You can vary the format and location of the claims.	<a href="#">JWTClaimsConfig</a>
<b>tokenLocation</b>	Specifies the source of the JWT authentication token and how it should be parsed.	<a href="#">JWTTokenLocation</a>
<b>key</b>	Configuration for the JWT key, specifying how the incoming JWT will be verified and decoded. In this example, we've used a fixed key that's stored as an environment variable in our root-level <code>.env</code> file that is also mapped to the <code>subgraph.yaml</code> in our <code>/globals</code> directory.	<a href="#">JWTKey</a>

## Webhook

An *AuthConfig* for *Webhook* mode:

```

kind: AuthConfig
version: v2

```

```

definition:
  mode:
    webhook:
      url: http://auth_hook:3050/validate-request
      method: Post

```

Field	Description	Reference
<code>url</code>	The URL of the authentication webhook that will be called to validate authentication.	<a href="#">AuthHookConfig</a>
<code>method</code>	The HTTP method ( <code>Get</code> or <code>Post</code> ) that will be used to make the request to the auth hook.	<a href="#">AuthHookMethod</a>

## Metadata structure

### AuthConfig

Definition of the authentication configuration used by the API server.

One of the following values:

Value	Description
<a href="#">AuthConfig1</a>	Definition of the authentication configuration v1, used by the API server.
<a href="#">AuthConfig2</a>	Definition of the authentication configuration v2, used by the API server.

### AuthConfig2

Definition of the authentication configuration v2, used by the API server.

Key	Value	Required	Description
<code>kind</code>	<code>AuthConfig</code>	true	

Key	Value	Required	Description
version	v2	true	
definition	AuthConfigV2	true	Definition of the authentication configuration v2, used by the API server.

## AuthConfigV2

Definition of the authentication configuration v2, used by the API server.

Key	Value	Required	Description
mode	AuthModeConfig	true	

## AuthConfig1

Definition of the authentication configuration v1, used by the API server.

Key	Value	Required	Description
kind	AuthConfig	true	
version	v1	true	
definition	AuthConfigV1	true	Definition of the authentication configuration v1, used by the API server.

## AuthConfigV1

Definition of the authentication configuration v1, used by the API server.

Key	Value	Required	Description
allowRoleEmulationBy	Role / null	false	
mode	AuthModeConfig	true	The configuration

Key	Value	Required	Description
			for the authentication mode to use - webhook, JWT or NoAuth.

## AuthModeConfig

The configuration for the authentication mode to use - webhook, JWT or NoAuth.

Must have exactly one of the following fields:

Key	Value	Required	Description
webhook	AuthHookConfig	false	The configuration of the authentication webhook.
jwt	JWTConfig	false	JWT config according to which the incoming JWT will be verified and decoded to extract the session variable claims.
noAuth	NoAuthConfig	false	Configuration used when running engine without authentication

## NoAuthConfig

Configuration used when running engine without authentication

Key	Value	Required	Description
role	Role	true	role to assume whilst running the engine
sessionVariables	SessionVariables	true	static session variables to use whilst running the engine

Example:

```
role: admin
sessionVariables:
  x-hasura-user-id: '100'
```

## SessionVariables

static session variables to use whilst running the engine

Key	Value	Required	Description
<customKey>	string	false	Value of a session variable

## JWTConfig

JWT config according to which the incoming JWT will be verified and decoded to extract the session variable claims.

Key	Value	Required	Description
audience	[string] / null	false	Optional validation to ensure that the <code>aud</code> field is a member of the <code>audiences</code> received, otherwise will throw error.
issuer	string / null	false	Optional validation to ensure that the <code>iss</code> field is a member of the <code>issuers</code> received, otherwise will throw error.
allowedSkew	integer / null	false	Allowed leeway (in seconds) to the <code>exp</code> validation account for clock skew.
claimsConfig	JWTClaimsConfig	true	Claims config. Either specified via <code>claims_mappings</code> or <code>claims_namespace</code> .

Key	Value	Required	Description
tokenLocation	JWTTokenLocation	true	Source of the JWT authentication token.
key	JWTKey	true	Mode according to which the JWT auth is configured

Example:

```

audience: null
issuer: null
allowedSkew: null
claimsConfig:
  namespace:
    claimsFormat: Json
    location: /claims.jwt.hasura.io
tokenLocation:
  type: BearerAuthorization
key:
  fixed:
    algorithm: HS256
    key:
      value: token

```

## JWTKey

JWT key configuration according to which the incoming JWT will be decoded.

Must have exactly one of the following fields:

Key	Value	Required	Description
fixed	JWTKeyConfig	false	JWT Secret config according to which the incoming JWT will be decoded.
jwkFromUrl	string	false	

## JWTKeyConfig

JWT Secret config according to which the incoming JWT will be decoded.

Key	Value	Required	Description
algorithm	JWTAlgorithm	true	The algorithm used to decode the JWT.
key	EnvironmentValue	true	The key to use for decoding the JWT.

## EnvironmentValue

Either a literal string or a reference to a Hasura secret

Must have exactly one of the following fields:

Key	Value	Required	Description
value	string	false	
valueFromEnv	string	false	

## JWTAlgorithm

The algorithm used to decode the JWT.

One of the following values:

Value	Description
HS256	HMAC using SHA-256
HS384	HMAC using SHA-384
HS512	HMAC using SHA-512
ES256	ECDSA using SHA-256
ES384	ECDSA using SHA-384
RS256	RSASSA-PKCS1-v1_5 using SHA-256

<b>Value</b>	<b>Description</b>
RS384	RSASSA-PKCS1-v1_5 using SHA-384
RS512	RSASSA-PKCS1-v1_5 using SHA-512
PS256	RSASSA-PSS using SHA-256
PS384	RSASSA-PSS using SHA-384
PS512	RSASSA-PSS using SHA-512
EdDSA	Edwards-curve Digital Signature Algorithm (EdDSA)

## JWTTokenLocation

Source of the Authorization token

One of the following values:

<b>Value</b>	<b>Description</b>
JWTBearerAuthorizationLocation	Get the bearer token from the <code>Authorization</code> header.
JWTCookieLocation	Get the token from the Cookie header under the specified cookie name.
JWTHeaderLocation	Custom header from where the header should be parsed from.

## JWTHeaderLocation

Custom header from where the header should be parsed from.

<b>Key</b>	<b>Value</b>	<b>Required</b>	<b>Description</b>
type	Header	true	
name	string	true	

## JWTCookieLocation

Get the token from the `Cookie` header under the specified cookie name.

Key	Value	Required	Description
<code>type</code>	<code>Cookie</code>	true	
<code>name</code>	string	true	

## JWTBearerAuthorizationLocation

Get the bearer token from the `Authorization` header.

Key	Value	Required	Description
<code>type</code>	<code>BearerAuthorization</code>	true	

## JWTCClaimsConfig

Config to describe how/where the engine should look for the claims within the decoded token.

Must have exactly one of the following fields:

Key	Value	Required	Description
<code>locations</code>	<code>JWTCClaimsMap</code>	false	Can be used when Hasura claims are not all present in the single object, but individual claims are provided a JSON pointer within the decoded JWT and optionally a default value.
<code>namespace</code>	<code>JWTCClaimsNamespace</code>	false	Used when all of the Hasura claims are present in a single object within the decoded JWT.

## JWTCClaimsNamespace

Used when all of the Hasura claims are present in a single object within the decoded JWT.

Key	Value	Required	Description
claimsFormat	JWTClaimsFormat	true	Format in which the Hasura claims will be present.
location	string	true	Pointer to lookup the Hasura claims within the decoded claims.

## JWTClaimsFormat

Format in which the Hasura claims will be present.

One of the following values:

Value	Description
Json	Claims will be in the JSON format.
StringifiedJson	Claims will be in the Stringified JSON format.

## JWTClaimsMap

Can be used when Hasura claims are not all present in the single object, but individual claims are provided a JSON pointer within the decoded JWT and optionally a default value.

Key	Value	Required	Description
x-hasura-default-role	JWTClaimsMappingEntry	true	JSON pointer to lookup the default role within the decoded JWT.
x-hasura-allowed-roles	JWTClaimsMappingEntry	true	JSON pointer to lookup the allowed roles within the decoded JWT.

## JWTCClaimsMappingEntry

JSON pointer to lookup the default role within the decoded JWT.

Must have exactly one of the following fields:

Key	Value	Required	Description
literal	Role	false	
path	JWTCClaimsMappingPathEntry	false	Entry to lookup the Hasura claims at the specified JSON Pointer

## JWTCClaimsMappingPathEntry

Entry to lookup the Hasura claims at the specified JSON Pointer

Key	Value	Required	Description
path	string	true	JSON pointer to find the particular claim in the decoded JWT token.
default	Role / null	false	Default value to be used when no value is found when looking up the value using the path.

## AuthHookConfig

The configuration of the authentication webhook.

Key	Value	Required	Description
url	string	true	The URL of the authentication webhook.
method	AuthHookMethod	true	The HTTP method to be used to make the request to the auth hook.

Example:

```
url: http://auth_hook:3050/validate-request  
method: Post
```

## AuthHookMethod

The HTTP method to be used to make the request to the auth hook.

**Value:**  /

## Role

**Value:** string

# Compatibility Config

## Introduction

The CompatibilityConfig object is a metadata object that defines the compatibility configuration of the Hasura metadata.

## How CompatibilityConfig works

### Lifecycle

By default, all projects are created with a default CompatibilityConfig object in the **globals** subgraph. It can be defined in any subgraph of your choice, but only once across your supergraph.

The `date` field in the CompatibilityConfig object specifies the date after which any backwards incompatible changes made to Hasura DDN won't impact the metadata.

## Examples

### Require valid NDC v01 version

Date: `2024-11-15`

**Description:** When enabled, raise an error when there is an invalid version in the capabilities of the data connector's schema. To fix the error, please consider upgrading to the latest version of the data connector. Before this, we would have assumed that they were NDC v0.1.x connectors.

### Require unique command GraphQL names

Date: `2024-10-07`

**Description:** When enabled, raise an error when there is a conflict with a GraphQL root field or GraphQL type name that is already in use. Before this, the command would be silently dropped from the GraphQL schema.

### Propagate boolean expression deprecation status

Date: 2024-09-26

**Description:** Enables propagating `deprecated` status for relationship fields to boolean expressions that use them. The default GraphQL introspection behavior is to hide such fields from the schema, so this behavior is opt-in to avoid breakage.

## Disallow scalar types names conflicting with inbuilt types

Date: 2024-09-26

**Description:** When enabled, defining a `ScalarType` metadata type that conflicts with any of the built-in type names (`ID`, `Int`, `String`, `Boolean`, `Float`) now raises an error that halts the build.

## Require nested array filtering capability

Date: 2024-09-18

**Description:** When enabled, defining a nested array field against a model whose underlying data connector does not have the `query.exists.nested_collections` capability defined will raise an error that halts the build.

## Bypass `relation_comparisons` NDC Capability

Date: 2024-09-03

**Description:** Enables bypassing the `relation_comparisons` NDC capability for relationship predicates. This allows you to use relationships in boolean expressions even if the data connector lacks compatibility. When the capability is available, relationship predicates are resolved directly within the native data connector for more efficient processing. If not, the predicates are handled at the API layer.

## Require GraphQL Config

Date: 2024-06-30

**Description:** Enforce the need for [GraphQL Config](#).

---

# Metadata structure

## v2\_CompatibilityConfig

The compatibility configuration of the Hasura metadata.

Key	Value	Required	Description
kind	CompatibilityConfig	true	
date	string	true	Any backwards incompatible changes made to Hasura DDN after this date won't impact the metadata.

# Engine Plugins

## Introduction

Engine plugins are a way to extend the functionality of the Hasura GraphQL Engine. Engine plugins are executed at various stages of the request lifecycle. Engine plugins can be used to modify the request, response, or to perform custom operations.

## How LifeCyclePluginHooks work

Lifecycle plugin hooks are not generated by default when a Hasura project is created. They need to be manually added to the metadata. To see a guide, check out [this page](#).

When you create a LifeCyclePluginHook, you define the configuration for the plugin in your metadata by passing information like the URL of the hosted plugin and its name. The plugin is then executed at the specified stage of the request lifecycle.

After creating a new LifeCyclePluginHook in your metadata, you'll need to [create a new build](#) using the CLI.

## Examples

*An example of a LifecyclePluginHook:*

```
kind: LifecyclePluginHook
version: v1
definition:
  name: cloudflare allowlist
  url:
    valueFromEnv: ALLOW_LIST_URL
  pre: parse
  config:
    request:
      headers:
        additional:
          hasura-m-auth:
            valueFromEnv: M_AUTH_KEY
  session: {}
  rawRequest:
```

```
query: {}
variables: {}
```

Field	Description	Reference
name	The name of the lifecycle plugin hook.	<a href="#">LifecyclePluginHookPreParse</a>
url	The URL to access the plugin.	<a href="#">LifecyclePluginHookPreParse</a>
pre	The stage of the request lifecycle.	<a href="#">LifecyclePluginHookPreParse</a>
config	Configuration for the plugin.	<a href="#">PreParse</a> or <a href="#">PreResponse</a>
config.request	The shape of the request object.	<a href="#">PreParse</a> or <a href="#">PreResponse</a>
config.request.headers	The headers that should be included for each request.	<a href="#">LifecyclePluginHookHeaders</a>
config.request.rawRequest	The configuration of the raw request, including any queries or variables.	<a href="#">RawRequestConfig</a>

## Metadata structure

### LifecyclePluginHook

Definition of a lifecycle plugin hook.

Key	Value	Required	Description
kind	LifecyclePluginHook	true	
version	v1	true	
definition	LifecyclePluginHookV1	true	Definition of a lifecycle plugin hook - version 1.

Example:

```

kind: LifecyclePluginHook
version: v1
definition:
  pre: parse
  name: test
  url:
    value: http://localhost:8080
  config:
    request:
      headers:
        additional:
          hasura-m-auth:
            value: zZkhKqFjqXR4g5MZCsJUZCnhCcoPyZ
      session: {}
    rawRequest:
      query: {}
      variables: {}

```

## LifecyclePluginHookV1

Definition of a lifecycle plugin hook - version 1.

One of the following values:

Value	Description
LifecyclePreParsePluginHook	Definition of a lifecycle plugin hook for the pre-parse stage.
LifecyclePreResponsePluginHook	Definition of a lifecycle plugin hook for the pre-response stage.

## LifecyclePreResponsePluginHook

Definition of a lifecycle plugin hook for the pre-response stage.

Key	Value	Required	Description
pre	response	true	
name	string	true	The name of the lifecycle plugin hook.
url	EnvironmentValue	true	The URL to access the lifecycle plugin hook.
config	LifecyclePreResponsePluginHookConfig	true	Configuration for the lifecycle plugin hook.

## LifecyclePreResponsePluginHookConfig

Configuration for a lifecycle plugin hook.

Key	Value	Required	Description
request	LifecyclePreResponsePluginHookConfigRequest	true	Configuration for the lifecycle plugin hook request.

## LifecyclePreResponsePluginHookConfigRequest

Configuration for a lifecycle plugin hook request.

Key	Value	Required	Description
headers	<a href="#">LifecyclePluginHookHeadersConfig</a> / null	false	Configuration for the headers.
session	<a href="#">LeafConfig</a> / null	false	Configuration for the session (includes roles and session variables)
rawRequest	<a href="#">RawRequestConfig</a>	true	Configuration for the raw request.
rawResponse	<a href="#">LeafConfig</a> / null	false	Configuration for the response

## LifecyclePreParsePluginHook

Definition of a lifecycle plugin hook for the pre-parse stage.

Key	Value	Required	Description
pre	parse	true	
name	string	true	The name of the lifecycle plugin hook.
url	<a href="#">EnvironmentValue</a>	true	The URL to access the lifecycle plugin hook.
config	<a href="#">LifecyclePreParsePluginHookConfig</a>	true	Configuration for the lifecycle plugin hook.

## LifecyclePreParsePluginHookConfig

Configuration for a lifecycle plugin hook.

Key	Value	Required	Description
request	LifecyclePreParsePluginHookConfigRequest	true	Configuration for the request for the lifecycle plugin.

## LifecyclePreParsePluginHookConfigRequest

Configuration for a lifecycle plugin hook request.

Key	Value	Required	Description
headers	LifecyclePluginHookHeadersConfig / null	false	Configuration for the headers.
session	LeafConfig / null	false	Configuration for the session (includes roles and session variables)
rawRequest	RawRequestConfig	true	Configuration for the raw request.

## RawRequestConfig

Configuration for the raw request.

Key	Value	Required	Description
query	LeafConfig / null	false	Configuration for the query.

Key	Value	Required	Description
variables	LeafConfig / null	false	Configuration for the variables.

## LeafConfig

Leaf Configuration.

Key	Value	Required	Description
-----	-------	----------	-------------

## LifecyclePluginHookHeadersConfig

Configuration for a lifecycle plugin hook headers.

Key	Value	Required	Description
additional	HttpHeaders / null	false	Additional headers to be sent with the request.
forward	[string]	false	Headers to be forwarded from the incoming request.

## HttpHeaders

Key value map of HTTP headers to be sent with an HTTP request. The key is the header name and the value is a potential reference to an environment variable.

Key	Value	Required	Description
<customKey>	EnvironmentValue	false	

## EnvironmentValue

Either a literal string or a reference to a Hasura secret

Must have exactly one of the following fields:

Key	Value	Required	Description
value	string	false	
valueFromEnv	string	false	

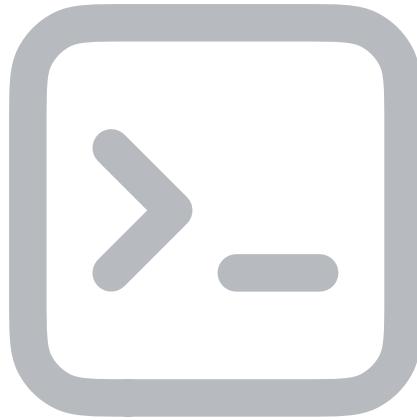
# Hasura CLI

Hasura introduces a new CLI, that gives you complete control over your data layer.

You can create projects, apply builds, and author metadata — all without leaving your favorite terminal.

## Quick Links

- [Learn how install the new CLI.](#)
- [Check out the commands.](#)



### INSTALL THE CLI FOR MACOS

Click here to learn how to install the CLI.

[Learn more >](#)

### INSTALL THE CLI FOR LINUX

Click here to learn how to install the CLI.

[Learn more >](#)

### INSTALL THE CLI FOR WINDOWS

Click here to learn how to install the CLI.

[Learn more >](#)



# Installation

Download the CLI binary below. Please follow the instructions for your system.

macOS and Linux

Windows

Simply run the installer script in your terminal:

```
curl -L https://graphql-engine-cdn.hasura.io/ddn/cli/v4/get.sh | bash
```

## Verify Installation

Running `ddn version` should print the CLI version, For example:

```
DDN CLI Version: v2.0.1
```



# DDN CLI: Commands

## Overview

The DDN CLI utilizes a typical command / flag syntax enabling you to quickly manage your supergraph. A typical command structure will follow this pattern:

```
ddn <command> <subcommand> <argument> --<flag> "<flag_value>"
```

 **NOTE**

A complete list of all CLI commands, including details and examples, can be found in the side navigation to the left under the Commands heading.

## Environment variables

You can configure additional behavior for the CLI using environment variables.

### **HASURA\_DDN\_PROJECT\_DIRECTORY**

Set the `HASURA_DDN_PROJECT_DIRECTORY` environment variable to the path of your project directory, which contains the `hasura.yaml` file. This is helpful when scripting automations and ensures that any scripts or commands run from different locations will automatically reference the correct project configuration.

## Getting help

You can use the `--help` flag to get information on any command.

For example, running `ddn --help` will return information on all available operations and flags:

```
DDDDDDDD\    DDDDDDDD\    NN\    NN\  
DD  __DD\  DD  __DD\  NNN\  NN |  
DD |  DD |  DD |  DD |  NNNN\  NN |
```

```
DD | DD | DD | DD | NN NN\NN |
DD | DD | DD | DD | NN \NNNN |
DD | DD | DD | DD | NN |\NNN |
DDDDDDDD | DDDDDDDD | NN | \NN |
\_____| \_____| \_| \_|

Usage:
  ddn [flags]
  ddn [command]

DDN operations
  project      Manage Hasura DDN Project

Metadata operations
  command      Perform Command related operations
  connector    Perform Connector related operations
  connector-link Perform DataConnectorLink related operations
  model        Perform Model related operations
  relationship Perform Relationship related operations
  subgraph     Perform Subgraph related operations
  supergraph   Perform Supergraph related operations

Authentication operations
  auth         Manage Hasura DDN CLI Auth

Other operations
  codemod      Perform transformations on your Hasura project
  directory
  completion   Generate the autocompletion script for the
  specified shell
  console      Open the DDN console
  context      Perform context operations
  help         Help about any command
  plugins      Manage plugins for the CLI
  run          Run specific script from project's context
  config
    update-cli  Update this CLI to the latest version or to a
    specific version
    version     Prints the CLI version

Flags:
  -h, --help            help for ddn
  --log-level string    Log level. Can be DEBUG, WARN, INFO,
  ERROR, or FATAL. (default "INFO")
  --no-prompt           Do not prompt for required but
  missing flags
  --out string          Output format. Can be table, json or
  yaml. (default "table")
```

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)  
-v, --version      Prints the CLI version  
Use "ddn [command] --help" for more information about a  
command.
```

# DDN CLI: ddn

DDN Command Line Interface.

## Synopsis

```
DDDDDDD\  DDDDDDD\  NN\  NN\  
DD  __DD\  DD  __DD\  NNN\  NN |  
DD |  DD |  DD |  DD |  NNNN\  NN |  
DD |  DD |  DD |  DD |  NN  NN\NN |  
DD |  DD |  DD |  DD |  NN \NNNN |  
DD |  DD |  DD |  DD |  NN | \NNN |  
DDDDDDD |  DDDDDDD |  NN |  \NN |  
\_____|  \_____|  \_|  \|_ |
```

```
ddn [flags]
```

## Available operations

- [ddn auth](#) - Manage Hasura DDN CLI Auth
- [ddn codemod](#) - Perform transformations on your Hasura project directory
- [ddn command](#) - Perform Command related operations
- [ddn connector](#) - Perform Connector related operations
- [ddn connector-link](#) - Perform DataConnectorLink related operations
- [ddn console](#) - Open the DDN console
- [ddn context](#) - Perform context operations
- [ddn doctor](#) - Check if the dependencies of DDN CLI are present
- [ddn model](#) - Perform Model related operations
- [ddn plugins](#) - Manage plugins for the CLI
- [ddn project](#) - Manage Hasura DDN Project
- [ddn relationship](#) - Perform Relationship related operations
- [ddn run](#) - Run specific script from project's context config
- [ddn subgraph](#) - Perform Subgraph related operations
- [ddn supergraph](#) - Perform Supergraph related operations
- [ddn update-cli](#) - Update this CLI to the latest version or to a specific version

## Options

```
-h, --help                  help for ddn
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt                Do not prompt for required but missing
flags
--out string                Output format. Can be table, json or
yaml. (default "table")
--timeout int                Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

# DDN CLI: ddn auth

Manage Hasura DDN CLI Auth.

## Synopsis

Manage Hasura DDN CLI Auth

## Available operations

- [ddn auth login](#) - Login to DDN
- [ddn auth logout](#) - Logout from DDN
- [ddn auth print-pat](#) - Prints the PAT to STDOUT

## Options

```
-h, --help    help for auth
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn auth login

Login to DDN.

## Synopsis

Login to DDN

```
ddn auth login [flags]
```

## Examples

```
# Login with browser
ddn auth login
# Login with Personal Access Token
ddn auth login --pat <your-personal-access-token>
```

## Options

```
-h, --help            help for login
--pat string        Personal Access token [env: HASURA_DDN_PAT]
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn auth](#) - Manage Hasura DDN CLI Auth

# DDN CLI: ddn auth logout

Logout from DDN.

## Synopsis

Logout from DDN

```
ddn auth logout [flags]
```

## Examples

```
# Logout from DDN CLI
ddn auth logout
```

## Options

```
-h, --help    help for logout
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn auth](#) - Manage Hasura DDN CLI Auth



# DDN CLI: ddn auth print-pat

Prints the PAT to STDOUT.

## Synopsis

Prints the PAT to STDOUT

```
ddn auth print-pat [flags]
```

Alias: pp

## Examples

```
# Print the PAT as a string to STDOUT
ddn auth print-pat
# Print PAT as a JSON to STDOUT
ddn auth print-pat --out json
```

## Options

```
-h, --help    help for print-pat
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn auth](#) - Manage Hasura DDN CLI Auth

# DDN CLI: ddn codemod

Perform transformations on your Hasura project directory.

## Synopsis

Perform transformations on your Hasura project directory

## Available operations

- [ddn codemod fix-traces-env-var](#) - Fix env var used for configuring traces for connectors
- [ddn codemod rename-graphql-prefixes](#) - Rename GraphQL root field and type name prefixes in metadata
- [ddn codemod upgrade-context-v2-to-v3](#) - Upgrade project's context config from v2 to v3
- [ddn codemod upgrade-graphqlconfig-aggregate](#) - Upgrade GraphqlConfig to support aggregates
- [ddn codemod upgrade-object-boolean-expression-types](#) - Upgrade object boolean expression types metadata
- [ddn codemod upgrade-project-config-v2-to-v3](#) - Upgrade project directory from version v2 to v3
- [ddn codemod upgrade-supergraph-config-v1-to-v2](#) - Upgrade all Supergraph config files at the root of the project directory from v1 to v2

## Options

```
-h, --help    help for codemod
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
```

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface

# DDN CLI: ddn codemod fix-traces-env-var

Fix env var used for configuring traces for connectors.

## Synopsis

Fix env var used for configuring traces for connectors

```
ddn codemod fix-traces-env-var --dir <project-dir> [flags]
```

## Examples

```
# Fix env var used for configuring traces for connector
ddn codemod fix-traces-env-var --dir .
```

## Options

```
--dir string    The Hasura project directory (required)
-h, --help       help for fix-traces-env-var
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn codemod](#) - Perform transformations on your Hasura project directory



# DDN CLI: ddn codemod rename-graphql-prefixes

Rename GraphQL root field and type name prefixes in metadata.

## Synopsis

Rename GraphQL root field and type name prefixes in metadata. The from prefix will be stripped if provided, and the new prefix will be added. If the new prefix is already present, it will not be reapplied.

By default, subgraph.yaml is updated with the new prefixes.

```
ddn codemod rename-graphql-prefixes [flags]
```

## Examples

```
# Add root field and type name prefixes to the subgraph set in
# the context
ddn codemod rename-graphql-prefixes --graphql-root-field
'app_' --graphql-type-name 'App_'
# Change the root field prefix for the specified subgraph
# without modifying subgraph.yaml
ddn codemod rename-graphql-prefixes --subgraph
app/subgraph.yaml --graphql-root-field 'foo_' --from-graphql-
root-field 'app_' --no-update-subgraph-config
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
--from-graphql-root-field string	The previous GraphQL root field prefix
--from-graphql-type-name string	The previous GraphQL type name prefix

--graphql-root-field string	The new GraphQL root field prefix
--graphql-type-name string	The new GraphQL type name prefix
-h, --help	help for rename-graphql-prefixes
--no-update-subgraph-config	Do not update the subgraph config with the new prefixes
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt          Do not prompt for required but missing flags
--out string         Output format. Can be table, json or yaml. (default "table")
--timeout int        Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn codemod](#) - Perform transformations on your Hasura project directory

# DDN CLI: ddn codemod upgrade-context-v2-to-v3

Upgrade project's context config from v2 to v3.

## Synopsis

Upgrade project's context config from v2 to v3

```
ddn codemod upgrade-context-v2-to-v3 --dir <project-dir>  
[flags]
```

## Examples

```
# Upgrade context present in the current Hasura directory from  
v2 to v3  
ddn codemod upgrade-context-v2-to-v3 --dir .
```

## Options

```
--dir string      The Hasura project directory (required)  
-h, --help        help for upgrade-context-v2-to-v3
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt           Do not prompt for required but missing  
flags  
--out string          Output format. Can be table, json or yaml.  
(default "table")  
--timeout int         Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn codemod](#) - Perform transformations on your Hasura project directory

# DDN CLI: ddn codemod upgrade-graphqlconfig-aggregate

Upgrade GraphqlConfig to support aggregates.

## Synopsis

Upgrade GraphqlConfig to support aggregates

```
ddn codemod upgrade-graphqlconfig-aggregate [flags]
```

## Examples

```
# Run on the supergraph defined in the context
ddn codemod upgrade-graphqlconfig-aggregate
# Run on a specific supergraph
ddn codemod upgrade-graphqlconfig-aggregate --supergraph
./supergraph.cloud.yaml
# Run on a specific subgraph
ddn codemod upgrade-graphqlconfig-aggregate --subgraph
app/subgraph.yaml
```

## Options

--ci	Disables the use of context
-c, --context string <current_context>	Name of the context to use. (default
-h, --help	help for upgrade-graphqlconfig-
aggregate	
--subgraph string	Path to Subgraph config file
--supergraph string	Path to Supergraph config file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn codemod](#) - Perform transformations on your Hasura project directory

# DDN CLI: ddn codemod upgrade-object-boolean-expression-types

Upgrade object boolean expression types metadata.

## Synopsis

Upgrade object boolean expression types metadata

```
ddn codemod upgrade-object-boolean-expression-types [flags]
```

## Examples

```
# Run on the supergraph defined in the context
ddn codemod upgrade-object-boolean-expression-types
# Run on a specific supergraph
ddn codemod upgrade-object-boolean-expression-types --supergraph ./supergraph.cloud.yaml
# Run on a specific subgraph
ddn codemod upgrade-object-boolean-expression-types --subgraph app/subgraph.yaml
```

## Options

--ci	Disables the use of context
-c, --context string <current_context>	Name of the context to use. (default
-h, --help	help for upgrade-object-boolean-expression-types
--subgraph string	Path to Subgraph config file
--supergraph string	Path to Supergraph config file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn codemod](#) - Perform transformations on your Hasura project directory

# DDN CLI: ddn codemod upgrade-project-config-v2-to-v3

Upgrade project directory from version v2 to v3.

## Synopsis

Upgrade project directory from version v2 to v3

```
ddn codemod upgrade-project-config-v2-to-v3 --dir <project-dir>  
[flags]
```

## Examples

```
# Upgrade project in the current directory from v2 to v3  
ddn codemod upgrade-project-config-v2-to-v3 --dir .
```

## Options

```
--dir string      The Hasura project directory (required)  
-h, --help        help for upgrade-project-config-v2-to-v3
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt           Do not prompt for required but missing  
flags  
--out string          Output format. Can be table, json or yaml.  
(default "table")  
--timeout int         Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn codemod](#) - Perform transformations on your Hasura project directory

# DDN CLI: ddn codemod upgrade-supergraph-config-v1-to-v2

Upgrade all Supergraph config files at the root of the project directory from v1 to v2.

## Synopsis

Upgrade all Supergraph config files at the root of the project directory from v1 to v2

```
ddn codemod upgrade-supergraph-config-v1-to-v2 --dir <project-dir> [flags]
```

## Examples

```
# Upgrade all Supergraph config files in the current directory
# from v1 to v2
ddn codemod upgrade-supergraph-config-v1-to-v2 --dir .
```

## Options

```
--dir string      The Hasura project directory (required)
-h, --help         help for upgrade-supergraph-config-v1-to-v2
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int          Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn codemod](#) - Perform transformations on your Hasura project directory

# DDN CLI: ddn command

Perform Command related operations.

## Synopsis

Perform Command related operations

Alias: commands

## Available operations

- [ddn command add](#) - Add new Commands to the local metadata
- [ddn command update](#) - Update Commands in the local metadata

## Options

```
-h, --help    help for command
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn command add

Add new Commands to the local metadata.

## Synopsis

Add new Commands to the local metadata

```
ddn command add <connector-link-name> <procedure/function-name>  
[flags]
```

## Examples

```
# Add all Commands for DataConnectorLink "myfns" in the  
subgraph set in the context  
ddn command add myfns "*"  
# Add a new Command "Login" from the procedure "Login" in the  
DataConnectorLink "myfns" in "app" Subgraph  
ddn command add myfns Login --subgraph ./app/subgraph.yaml  
# Add all the Commands from the procedures/functions in the  
DataConnectorLink "myfns" in "app" Subgraph  
ddn command add myfns "*" --subgraph ./app/subgraph.yaml  
# Add Commands filtered by glob pattern from the  
procedures/functions in the DataConnectorLink "myfns" in "app"  
Subgraph  
ddn command add myfns user* --subgraph ./app/subgraph.yaml
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for add
--pattern string	Pattern to detect targets. Can be 'glob' or 'literal'. (default "glob")
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn command](#) - Perform Command related operations

# DDN CLI: ddn command update

Update Commands in the local metadata.

## Synopsis

Update Commands in the local metadata

```
ddn command update <command-name> [flags]
```

## Examples

```
# Update all Commands using the subgraph set in current context
ddn command update "*"
# Update the Command "Login" in the "app" Subgraph
ddn command update Login --subgraph ./app/subgraph.yaml
# Update all the Commands in "app" Subgraph
ddn command update "*" --subgraph ./app/subgraph.yaml
# Update Commands filtered by glob pattern in the Subgraph
"app"
ddn command update user* --subgraph ./app/subgraph.yaml
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for update
--pattern string	Pattern to detect targets. Can be 'glob' or 'literal'. (default "glob")
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn command](#) - Perform Command related operations

# DDN CLI: ddn connector

Perform Connector related operations.

## Synopsis

Perform Connector related operations

Alias: connectors

## Available operations

- [ddn connector build](#) - Perform ConnectorBuild related operations
- [ddn connector init](#) - Add a new Connector
- [ddn connector introspect](#) - Introspect the Connector data source to update the Connector configuration
- [ddn connector plugin](#) - Run a subcommand from a Connector plugin
- [ddn connector setenv](#) - Run specified command with environment variables set

## Options

```
-h, --help    help for connector
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn connector build

Perform ConnectorBuild related operations.

## Synopsis

Perform ConnectorBuild related operations

## Available operations

- [ddn connector build create](#) - Create a ConnectorBuild on Hasura DDN
- [ddn connector build delete](#) - Delete a ConnectorBuild from a Project
- [ddn connector build get](#) - List ConnectorBuilds or get details of a specific one from Hasura DDN
- [ddn connector build logs](#) - Get logs of a ConnectorBuild from Hasura DDN

## Options

```
-h, --help    help for build
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector](#) - Perform Connector related operations



# DDN CLI: ddn connector build create

Create a ConnectorBuild on Hasura DDN.

## Synopsis

Create a ConnectorBuild on Hasura DDN

```
ddn connector build create --connector <path-to-connector-  
config-file> [flags]
```

## Options

--ci	Disables the use of context
--connector string	Path to Connector config file
-c, --context string	Name of the context to use. (default <current_context>)
-e, --env stringArray	Environment variable, e.g. key=val. Can be repeated to provide multiple env vars
--env-file stringArray	Path to .env file. Can be repeated to provide multiple env files
-h, --help	help for create
-p, --project string	DDN Project name
--target-connector-link string	DataConnectorLink to update with the schema from the ConnectorBuild
--target-env-file string	Path to the env file in which the Build URLs should be updated in
--target-subgraph string	Path to Subgraph config file containing target DataConnectorLink
--update-connector-link-schema	Update DataConnectorLink schema with the NDC schema of the built connector. (default: false)

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector build](#) - Perform ConnectorBuild related operations

# DDN CLI: ddn connector build delete

Delete a ConnectorBuild from a Project.

## Synopsis

Delete a ConnectorBuild from a Project

```
ddn connector build delete <connector-build-id> [flags]
```

## Examples

```
# Delete a ConnectorBuild
ddn connector build delete 1d4f4831-54a2-4ded-b680-
07d00510a522
```

## Options

```
-h, --help    help for delete
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector build](#) - Perform ConnectorBuild related operations



# DDN CLI: ddn connector build get

List ConnectorBuilds or get details of a specific one from Hasura DDN.

## Synopsis

List ConnectorBuilds or get details of a specific one from Hasura DDN

```
ddn connector build get [connector-build-id] [flags]
```

## Examples

```
# Get details of a ConnectorBuild
ddn connector build get 1d4f4831-54a2-4ded-b680-07d00510a522
# Get details of all ConnectorBuilds for Connector "mydb" for a
Project and Subgraph
ddn connector build get --connector-name mydb --project
myproject --subgraph-name myapp
# Get details of all ConnectorBuilds for Connector defined in a
Connector config file
ddn connector build get --connector
./myapp/connector/connector.cloud.yaml
# Get details of all ConnectorBuilds for a Project
ddn connector build get --project myproject
# Get details of all ConnectorBuilds for a Subgraph in a
Project
ddn connector build get --project myproject --subgraph-name
myapp
# Get NDC Schema of a ConnectorBuild
ddn connector build get 1d4f4831-54a2-4ded-b680-07d00510a522 -
-schema
```

## Options

--ci	Disables the use of context
--connector string	Path to Connector config file

--connector-name string	Connector name
-c, --context string (default <current_context>)	Name of the context to use.
-h, --help	help for get
-p, --project string	DDN Project name
--schema	Get NDC schema of ConnectorBuild
--subgraph-name string	Subgraph name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string (default "table")	Output format. Can be table, json or yaml.
--timeout int	Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)

## Parent operation

- [ddn connector build](#) - Perform ConnectorBuild related operations

# DDN CLI: ddn connector build logs

Get logs of a ConnectorBuild from Hasura DDN.

## Synopsis

Get logs of a ConnectorBuild from Hasura DDN

```
ddn connector build logs <connector-build-id> [flags]
```

## Examples

```
# Get running deploy logs
ddn connector build logs <connector-build-id>
# Get running deploy logs and keep following
ddn connector build logs <connector-build-id> --follow
# Get running deploy logs and keep following since a specified
# time duration
ddn connector build logs <connector-build-id> --follow --since
10m
# Get build logs of a ConnectorBuild
ddn connector build logs <connector-build-id> --build
```

## Options

--build (default: false)	Specifies whether to show the build logs.
--follow	Specifies whether to continuously follow the deployment logs. (default: false)
-h, --help	help for logs
--since string	Specifies the starting time for log retrieval. Can be in ISO format (e.g. 2024-03-26T11:05:15Z) or a duration (e.g. 5m for 5 minutes ago). (By default, prints the entire available logs).

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector build](#) - Perform ConnectorBuild related operations

# DDN CLI: ddn connector init

Add a new Connector.

## Synopsis

Add a new Connector

```
ddn connector init [connector-name] --hub-connector <connector-type> [flags]
```

## Examples

```
# Initialize a Connector interactively in a step by step manner
ddn connector init -i
# Initialize a Postgres Connector "mydb" in the Subgraph "app"
ddn connector init mydb --subgraph ./app/subgraph.yaml --hub-connector hasura/postgres
# Initialize a Postgres Connector "mydb" inside the directory
./connector
ddn connector init mydb --dir ./connector --hub-connector hasura/postgres
# Initialize a NodeJS Connector "mylambda" in the Subgraph
"app" on port 8765
ddn connector init mylambda --subgraph ./app/subgraph.yaml --hub-connector hasura/nodejs --configure-port 8765
```

## Options

--add-to-compose-file string	The compose file to include the generated connector compose file
--ci	Disables the use of context
--configure-port string	Initialize the connector with an already configured port
-c, --context string (default <current_context>)	Name of the context to use.

--dir string	Directory to initialize the Connector
-h, --help	help for init
--hub-connector string	Name and version of Connector in Hasura Connector Hub. ref: <a href="https://hasura.io/connectors">https://hasura.io/connectors</a>
-i, --interactive	Interactive mode
--no-link	Do not create a ConnectorLink
--subgraph string	Subgraph to initialize the Connector in
--target-env-file string	Path to the environment file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt          Do not prompt for required but missing flags
--out string         Output format. Can be table, json or yaml. (default "table")
--timeout int        Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector](#) - Perform Connector related operations

# DDN CLI: ddn connector introspect

Introspect the Connector data source to update the Connector configuration.

## Synopsis

Introspect the Connector data source to update the Connector configuration

```
ddn connector introspect <connector-name> --subgraph <path-to-subgraph-config-file> [flags]
```

## Examples

```
# Introspect Connector my_db from Subgraph located at
# ./foo/subgraph.yaml and update DataConnectorLink
ddn connector introspect my_db --subgraph ./foo/subgraph.yaml
# Introspect Connector located at ./foo/my_db/connector.yaml
ddn connector introspect --connector
./foo/my_db/connector.yaml
# Introspect Connector my_db but do not update
DataConnectorLink
ddn connector introspect my_db --subgraph ./foo/subgraph.yaml
--no-update-link
```

## Options

--add-all-resources	Add all Models and Commands from the updated DataConnectorLink to the local metadata
--ci	Disables the use of context
--connector string	Path to Connector config file
-c, --context string	Name of the context to use. (default <current_context>)
-e, --env stringArray	Environment variable, e.g. key=val. Can be repeated to provide multiple env vars
--env-file stringArray	Path to .env file. Can be repeated to provide multiple env files

-h, --help	help for introspect
--no-update-link in the metadata	Ignore updating DataConnectorLink
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector](#) - Perform Connector related operations

# DDN CLI: ddn connector plugin

Run a subcommand from a Connector plugin.

## Synopsis

Run a subcommand from a Connector plugin

```
ddn connector plugin [flags]
```

## Options

--ci	Disables the use of context
--connector string	Path to Connector config file
-c, --context string	Name of the context to use. (default <current_context>)
-e, --env stringArray	Environment variable, e.g. key=val. Can be repeated to provide multiple env vars
--env-file stringArray	Path to .env file. Can be repeated to provide multiple env files
-h, --help	help for plugin

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")
--timeout int	Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)

## Parent operation

- [ddn connector](#) - Perform Connector related operations



# DDN CLI: ddn connector setenv

Run specified command with environment variables set.

## Synopsis

Run specified command with environment variables set

```
ddn connector setenv --connector <path-to-connector-config-file> -- <command> [flags]
```

## Examples

```
# Set environment variables for the Connector data source
ddn connector setenv --connector
./foo/my_db.connector.local.yaml -- npm run start
```

## Options

--ci	Disables the use of context
--connector string	Path to Connector config file
-c, --context string	Name of the context to use. (default <current_context>)
-e, --env stringArray	Environment variable, e.g. key=val. Can be repeated to provide multiple env vars
--env-file stringArray	Path to .env file. Can be repeated to provide multiple env files
-h, --help	help for setenv

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags

```
--out string          Output format. Can be table, json or yaml.  
(default "table")  
--timeout int         Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector](#) - Perform Connector related operations

# DDN CLI: ddn connector-link

Perform DataConnectorLink related operations.

## Synopsis

Perform DataConnectorLink related operations

Alias: connector-links

## Available operations

- [ddn connector-link add](#) - Add a new DataConnectorLink to a Subgraph
- [ddn connector-link add-resources](#) - Add all models, commands and relationships from a DataConnectorLink's schema
- [ddn connector-link show](#) - Show DataConnectorLink details
- [ddn connector-link update](#) - Fetch NDC details from the Connector and update the DataConnectorLink

## Options

```
-h, --help    help for connector-link
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn connector-link add

Add a new DataConnectorLink to a Subgraph.

## Synopsis

Add a new DataConnectorLink to a Subgraph

```
ddn connector-link add <connector-link-name> [flags]
```

## Examples

```
# Add a DataConnectorLink to the Subgraph "app"
ddn connector-link add mydb --subgraph ./app/subgraph.yaml
# Add a DataConnectorLink to the Subgraph "app" and configure
its connector URL as the Connector's local Docker service URL
ddn connector-link add mydb --subgraph app/subgraph.yaml --configure-host http://local.hasura.dev:<port>
```

## Options

--ci	Disables the use of context
--configure-connector-token string	Token used to authenticate requests to the Connector
--configure-host string	Read and Write URL of the Connector
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for add
--subgraph string	Path to Subgraph config file
--target-env-file string	Subgraph env file to write the connector URLs to

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector-link](#) - Perform DataConnectorLink related operations

# DDN CLI: ddn connector-link add-resources

Add all models, commands and relationships from a DataConnectorLink's schema.

## Synopsis

Add all models, commands and relationships from a DataConnectorLink's schema

```
ddn connector-link add-resources <connector-link-name> [flags]
```

## Examples

```
# Add all models, commands and relationships from the schema of
# DataConnectorLink 'mydb' for Subgraph config 'app'
ddn connector-link add-resources mydb --subgraph
./app/subgraph.yaml
```

## Options

--ci	Disables the use of context
-c, --context string <current_context>	Name of the context to use. (default
-h, --help	help for add-resources
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string (default "table")	Output format. Can be table, json or yaml.

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector-link](#) - Perform DataConnectorLink related operations

# DDN CLI: ddn connector-link show

Show DataConnectorLink details.

## Synopsis

Show DataConnectorLink details

```
ddn connector-link show <connector-link-name> [flags]
```

## Examples

```
# Show DataConnectorLink details for `mydb`  
ddn connector-link show mydb
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for show
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")
--timeout int	Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)

## Parent operation

- [ddn connector-link](#) - Perform DataConnectorLink related operations

# DDN CLI: ddn connector-link update

Fetch NDC details from the Connector and update the DataConnectorLink.

## Synopsis

Fetch NDC details from the Connector and update the DataConnectorLink

```
ddn connector-link update <connector-link-name> [flags]
```

## Examples

```
# Update the schema of a DataConnectorLink 'mydb' for Subgraph config 'app'  
ddn connector-link update mydb --subgraph ./app/subgraph.yaml  
# Update the schema of a DataConnectorLink 'mydb' and add all Models and Commands to the metadata for Subgraph config 'app'  
ddn connector-link update mydb --add-all-resources --subgraph  
./app/subgraph.yaml
```

## Options

--add-all-resources	Add all Models and Commands from the updated DataConnectorLink to the local metadata
--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-e, --env stringArray	Environment variable, e.g. key=val. Can be repeated to provide multiple env vars
--env-file stringArray	Path to .env file. Can be repeated to provide multiple env files
-h, --help	help for update
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn connector-link](#) - Perform DataConnectorLink related operations

# DDN CLI: ddn console

Open the DDN console.

## Synopsis

Open the DDN console

```
ddn console [flags]
```

## Examples

```
# Open the console for the DDN project set in the context
ddn console
# Open the local dev console
ddn console --local
# Open the local dev console with a specific engine url
ddn console --url http://localhost:8080
# Open the console for a specific DDN project
ddn console --project my-project-123
# Open the console for a specific SupergraphBuild
ddn console --project my-project-123 --build-version build-
version-123
```

## Options

--build-version string	SupergraphBuild version
-h, --help	help for console
--local	Open the local dev console
-p, --project string	DDN Project name
--url string	Local engine url

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface

# DDN CLI: ddn context

Perform context operations.

## Synopsis

Manage default value of keys to be used in DDN CLI commands

## Available operations

- [ddn context create-context](#) - Create a new context
- [ddn context get](#) - Get the value of a key in the context
- [ddn context get-context](#) - List contexts or get details of a specific context
- [ddn context get-current-context](#) - Get name and contents of current context
- [ddn context set](#) - Set the value of a key in the context
- [ddn context set-current-context](#) - Set the current context
- [ddn context unset](#) - Unset the value of a key in the context

## Options

```
-h, --help    help for context
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn context create-context

Create a new context.

## Synopsis

Create a new context

```
ddn context create-context <name> [flags]
```

## Examples

```
# Create a new context called staging
ddn context create-context staging
```

## Options

```
-h, --help    help for create-context
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn context](#) - Perform context operations



# DDN CLI: ddn context get

Get the value of a key in the context.

## Synopsis

Get the value of a key in the context

```
ddn context get <key> (Allowed keys: localEnvFile,  
cloudEnvFile, selfHostedDataPlane, project, supergraph,  
subgraph) [flags]
```

## Examples

```
# Get the Project name set in the context  
ddn context get project  
# Get the Supergraph config file path set in the context  
ddn context get supergraph
```

## Options

```
-c, --context string    Name of the context to use. (default  
<current_context>)  
-h, --help               help for get
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt           Do not prompt for required but missing  
flags  
--out string          Output format. Can be table, json or yaml.  
(default "table")
```

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn context](#) - Perform context operations

# DDN CLI: ddn context get-context

List contexts or get details of a specific context.

## Synopsis

List contexts or get details of a specific context

```
ddn context get-context [name] [flags]
```

## Examples

```
# Get list of contexts
ddn context get-context
# Get details of context 'default'
ddn context get-context default
```

## Options

```
-h, --help    help for get-context
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn context](#) - Perform context operations

# DDN CLI: ddn context get-current-context

Get name and contents of current context.

## Synopsis

Get name and contents of current context

```
ddn context get-current-context [flags]
```

## Examples

```
# Get name and contents of current context
ddn context get-current-context
```

## Options

```
-h, --help    help for get-current-context
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn context](#) - Perform context operations



# DDN CLI: ddn context set

Set the value of a key in the context.

## Synopsis

Set default value of keys to be used in DDN CLI commands

```
ddn context set <key> <value> (Allowed keys: cloudEnvFile,  
selfHostedDataPlane, project, supergraph, subgraph,  
localEnvFile) [flags]
```

## Examples

```
# Set the Project name in the context  
ddn context set project foo-bar-1234  
# Set the local Supergraph config file path in the context  
ddn context set supergraph ./supergraph.local.yaml  
# Set the Subgraph config file path in the context  
ddn context set subgraph ./app/subgraph.yaml
```

## Options

```
-c, --context string    Name of the context to use. (default  
<current_context>)  
-h, --help               help for set
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt           Do not prompt for required but missing  
flags  
--out string          Output format. Can be table, json or yaml.  
(default "table")
```

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn context](#) - Perform context operations

# DDN CLI: ddn context set-current-context

Set the current context.

## Synopsis

Set the current context

```
ddn context set-current-context <name> [flags]
```

## Examples

```
# Set current context to 'default'  
ddn context set-current-context default
```

## Options

```
-h, --help    help for set-current-context
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn context](#) - Perform context operations



# DDN CLI: ddn context unset

Unset the value of a key in the context.

## Synopsis

Unset the value of a key in the context

```
ddn context unset <key> (Allowed keys: project, supergraph,  
subgraph, localEnvFile, cloudEnvFile, selfHostedDataPlane)  
[flags]
```

## Examples

```
# Unset the Project name set in the context  
ddn context unset project  
# Unset the Supergraph config file path set in the context  
ddn context unset supergraph
```

## Options

```
-c, --context string    Name of the context to use. (default  
<current_context>)  
-h, --help               help for unset
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt           Do not prompt for required but missing  
flags  
--out string          Output format. Can be table, json or yaml.  
(default "table")
```

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn context](#) - Perform context operations

# DDN CLI: ddn doctor

Check if the dependencies of DDN CLI are present.

## Synopsis

Check if the dependencies (Docker and Docker Compose) of DDN CLI are installed, are of the required version and if they are running.

```
ddn doctor [flags]
```

## Options

```
-h, --help    help for doctor
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn model

Perform Model related operations.

## Synopsis

Perform Model related operations

Alias: models

## Available operations

- [ddn model add](#) - Add new Models to the local metadata
- [ddn model update](#) - Update Models in the local metadata

## Options

```
-h, --help    help for model
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn model add

Add new Models to the local metadata.

## Synopsis

Add new Models to the local metadata

```
ddn model add <connector-link-name> <collection-name> [flags]
```

## Examples

```
# Add all Models for DataConnectorLink "mydb" in the subgraph
set in the context
ddn model add mydb "*"
# Add a new Model "Album" from the collection "Album" in the
DataConnectorLink "mydb" in Subgraph "app"
ddn model add mydb Album --subgraph ./app/subgraph.yaml
# Add all the Models from the collections in the
DataConnectorLink "mydb" in Subgraph "app"
ddn model add mydb "*" --subgraph ./app/subgraph.yaml
# Add Models filtered by glob pattern from the collections in
the DataConnectorLink "mydb" in Subgraph "app"
ddn model add mydb user* --subgraph ./app/subgraph.yaml
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for add
--pattern string	Pattern to detect targets. Can be 'glob' or 'literal'. (default "glob")
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn model](#) - Perform Model related operations

# DDN CLI: ddn model update

Update Models in the local metadata.

## Synopsis

Update Models in the local metadata

```
ddn model update <model-name> [flags]
```

## Examples

```
# Update all Models using the subgraph set in current context
ddn model update "*"
# Update the Model "Album" in the "app" Subgraph
ddn model update Album --subgraph ./app/subgraph.yaml
# Update all the Models in the Subgraph "app"
ddn model update "*" --subgraph ./app/subgraph.yaml
# Update Models filtered by glob pattern in the Subgraph "app"
ddn model update user* --subgraph ./app/subgraph.yaml
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for update
--pattern string	Pattern to detect targets. Can be 'glob' or 'literal'. (default "glob")
--subgraph string	Path to Subgraph config file

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--------------------	--

```
--no-prompt          Do not prompt for required but missing
flags
--out string         Output format. Can be table, json or yaml.
(default "table")
--timeout int        Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn model](#) - Perform Model related operations

# DDN CLI: ddn plugins

Manage plugins for the CLI.

## Synopsis

The functionality of the CLI can be extended by using plugins. For a list of all available plugins, run `ddn plugins list`, or visit this repository: <https://github.com/hasura/cli-plugins-index>.

If you're interested in contributing, please open a PR against this repo.

**Alias:** plugin

## Available operations

- `ddn plugins install` - Install a plugin from the index
- `ddn plugins list` - List all available plugins with index, versions and installation status
- `ddn plugins uninstall` - Uninstall a plugin
- `ddn plugins upgrade` - Upgrade a plugin to a newer version

## Options

```
-h, --help    help for plugins
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface

# DDN CLI: ddn plugins install

Install a plugin from the index.

## Synopsis

To install plugins that extend the functionality of the DDN CLI, you can use the `install` command. This command will install the plugin from the index and add it to your configuration file.

```
ddn plugins install <name> [flags]
```

Alias: `add`

## Examples

```
# Install a plugin named "ndc-postgres"  
ddn plugins install ndc-postgres
```

## Options

```
-h, --help           help for install  
--version string   Version to be installed
```

## Options inherited from parent operations

```
--log-level string   Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn plugins](#) - Manage plugins for the CLI

# DDN CLI: ddn plugins list

List all available plugins with index, versions and installation status.

## Synopsis

Run the plugins list command to see a list of all the available plugins which extend the functionality of the CLI, their versions and installation status.

```
ddn plugins list [flags]
```

**Alias:** ls

## Examples

```
# List all plugins
ddn plugins list
# List all plugins without updating the plugin index local
cache
ddn plugins list --dont-update-index
```

## Options

```
--dont-update-index  Don't update the plugin index local
cache, only show the list
-h, --help           help for list
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt          Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
```

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn plugins](#) - Manage plugins for the CLI

# DDN CLI: ddn plugins uninstall

Uninstall a plugin.

## Synopsis

To uninstall a plugin, run the `uninstall` command with the name of the plugin as an argument. If unsure of the plugin's name, you can run the `ddn plugins list` command to see a list of all the available plugins.

```
ddn plugins uninstall <plugin-name> [flags]
```

**Alias:** remove

## Examples

```
# Uninstall a plugin named "ndc-postgres"
ddn plugins uninstall ndc-postgres
```

## Options

```
-h, --help    help for uninstall
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn plugins](#) - Manage plugins for the CLI

# DDN CLI: ddn plugins upgrade

Upgrade a plugin to a newer version.

## Synopsis

To upgrade a plugin, run the upgrade command with the name of the plugin as an argument. If unsure of the plugin's name, you can run the `ddn plugins list` command to see a list of all the available plugins.

```
ddn plugins upgrade <plugin-name> [flags]
```

**Alias:** update

## Examples

```
# Upgrade a plugin "ndc-postgres" to a newer version
ddn plugins upgrade ndc-postgres
```

## Options

```
-h, --help            help for upgrade
--version string     Version to be upgraded
```

## Options inherited from parent operations

```
--log-level string   Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt          Do not prompt for required but missing
flags
--out string         Output format. Can be table, json or yaml.
(default "table")
--timeout int        Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn plugins](#) - Manage plugins for the CLI

# DDN CLI: ddn project

Manage Hasura DDN Project.

## Synopsis

Manage Hasura DDN Project

## Available operations

- [ddn project create](#) - Create a new Project on Hasura DDN
- [ddn project delete](#) - Delete a Project on Hasura DDN
- [ddn project get](#) - List Hasura DDN Projects or get details of a specific one
- [ddn project init](#) - Configure the local directory to use a Hasura DDN project, creating a DDN project and subgraphs as necessary
- [ddn project set-self-hosted-engine-url](#) - Set the engine's URL for a project in a self hosted data plane.
- [ddn project subgraph](#) - Manage Subgraphs in a Hasura DDN Project

## Options

```
-h, --help    help for project
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn project create

Create a new Project on Hasura DDN.

## Synopsis

Create a new Project on Hasura DDN

```
ddn project create [project-name] [flags]
```

## Examples

```
# Create a Project on Hasura DDN with auto-generated name
ddn project create
# Create a Project with name "test-project" on Hasura DDN
ddn project create test-project
```

## Options

```
--data-plane-id uuid    The DDN instance where the Project
should be hosted
-h, --help               help for create
--plan string            DDN Project plan
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn project](#) - Manage Hasura DDN Project

# DDN CLI: ddn project delete

Delete a Project on Hasura DDN.

## Synopsis

Delete a Project on Hasura DDN

```
ddn project delete <project-name> [flags]
```

## Examples

```
# Delete a Project "pet-lion-2649" on Hasura DDN
ddn project delete pet-lion-2649
```

## Options

```
-h, --help    help for delete
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn project](#) - Manage Hasura DDN Project



# DDN CLI: ddn project get

List Hasura DDN Projects or get details of a specific one.

## Synopsis

List Hasura DDN Projects or get details of a specific one

```
ddn project get [project-name] [flags]
```

## Examples

```
# List all your Projects on Hasura DDN.  
ddn project get  
# Get details of a Project "pet-lion-2649" on Hasura DDN.  
ddn project get pet-lion-2649
```

## Options

```
-h, --help    help for get
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt           Do not prompt for required but missing  
flags  
--out string          Output format. Can be table, json or yaml.  
(default "table")  
--timeout int         Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn project](#) - Manage Hasura DDN Project

# DDN CLI: ddn project init

Configure the local directory to use a Hasura DDN project, creating a DDN project and subgraphs as necessary.

## Synopsis

Configure the local directory to use a Hasura DDN project, creating a DDN project and subgraphs as necessary

```
ddn project init [project-name] [flags]
```

## Examples

```
# Initialize a new Hasura DDN project (with an auto-generated name) with subgraphs based on your local directory
ddn project init
# Configure the local directory to use an existing Hasura DDN project creating subgraphs on the DDN project as necessary
ddn project init --with-project myproject
```

## Options

--ci	Disables the use of context
-c, --context string (default <current_context>)	Name of the context to use.
--data-plane-id uuid	The DDN instance where the Project should be hosted
--env-file-name string	Env file to be created and added to context as cloudEnvFile (default ".env.cloud")
--from-env-file string	Env file to initialize the cloudEnvFile from
-h, --help	help for init
--plan string	DDN Project plan
--supergraph string	Path to Supergraph config file

```
--with-project string      Use an existing project instead of  
creating a new one
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn project](#) - Manage Hasura DDN Project

# DDN CLI: ddn project set-self-hosted-engine-url

Set the engine's URL for a project in a self hosted data plane..

## Synopsis

Set the engine's URL for a project in a self hosted data plane.

```
ddn project set-self-hosted-engine-url <url> [flags]
```

## Examples

```
# Set project URL to "example.com:3000" for project "pet-lion-2649"  
ddn project set-self-hosted-engine-url example.com:3000 --  
project pet-lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for set-self-hosted-engine-url
-p, --project string	DDN Project name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn project](#) - Manage Hasura DDN Project

# DDN CLI: ddn project subgraph

Manage Subgraphs in a Hasura DDN Project.

## Synopsis

Manage Subgraphs in a Hasura DDN Project

## Available operations

- [ddn project subgraph create](#) - Create a new Subgraph in a Hasura DDN Project
- [ddn project subgraph delete](#) - Delete a Subgraph from a Project
- [ddn project subgraph get](#) - List Subgraphs for a Hasura DDN Project or get details of a specific one

## Options

```
-h, --help    help for subgraph
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn project](#) - Manage Hasura DDN Project



# DDN CLI: ddn project subgraph create

Create a new Subgraph in a Hasura DDN Project.

## Synopsis

Create a new Subgraph in a Hasura DDN Project

```
ddn project subgraph create <subgraph-name> [flags]
```

## Examples

```
# Create a new Subgraph "app" in a Project
ddn project subgraph create app --project pet-lion-2649
# Create a new Subgraph "app" in a Project with a description
  ddn project subgraph create app --project pet-lion-2649 --
description "application management APIs"
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-d, --description string	(Optional) description of the subgraph
-h, --help	help for create
-p, --project string	DDN Project name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml.

```
(default "table")
--timeout int      Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn project subgraph](#) - Manage Subgraphs in a Hasura DDN Project

# DDN CLI: ddn project subgraph delete

Delete a Subgraph from a Project.

## Synopsis

Delete a Subgraph from a Project

```
ddn project subgraph delete <subgraph-name> [flags]
```

## Examples

```
# Delete a Subgraph 'app' from a Project
ddn project subgraph delete app --project pet-lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for delete
-p, --project string	DDN Project name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")
--timeout int	Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)

## Parent operation

- [ddn project subgraph](#) - Manage Subgraphs in a Hasura DDN Project

# DDN CLI: ddn project subgraph get

List Subgraphs for a Hasura DDN Project or get details of a specific one.

## Synopsis

List Subgraphs for a Hasura DDN Project or get details of a specific one

```
ddn project subgraph get [subgraph-name] [flags]
```

## Examples

```
# List all Subgraphs for a Project
ddn project subgraph get --project pet-lion-2649
# View details of a Subgraph "app" in a Project
ddn project subgraph get app --project pet-lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for get
-p, --project string	DDN Project name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn project subgraph](#) - Manage Subgraphs in a Hasura DDN Project

# DDN CLI: ddn relationship

Perform Relationship related operations.

## Synopsis

Perform Relationship related operations

Alias: relationships

## Available operations

- [ddn relationship add](#) - Adds Relationships from foreign keys on collection-name or targeting collection-name

## Options

```
-h, --help    help for relationship
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn relationship add

Adds Relationships from foreign keys on collection-name or targeting collection-name.

## Synopsis

Adds Relationships from foreign keys on collection-name or targeting collection-name

```
ddn relationship add <connector-link-name> <collection-name>  
[flags]
```

## Examples

```
# Add all Relationships for DataConnectorLink "mydb" in the  
subgraph set in the context  
ddn relationship add mydb "*"  
# Add Relationships for the collection "Album" in the  
DataConnectorLink "mydb" in the Subgraph "app"  
ddn relationship add mydb Album --subgraph ./app/subgraph.yaml  
# Add Relationships for collections that match the glob pattern  
"sales_*"  
ddn relationship add mydb "sales_*"  
# Add Relationships for the collection "Album" defined by the  
foreign key "artists_album_id_fkey" on the collection "Artist"  
ddn relationship add mydb Album --fk-collection Artist --fk-  
name artists_album_id_fkey
```

## Options

--ci	Disables the use of context
-c, --context string (default <current_context>)	Name of the context to use.
--fk-collection string on this collection	Only consider foreign keys defined on this collection
--fk-name string this name	Only consider foreign keys with this name

```
-h, --help                  help for add
--pattern string            Pattern to detect targets. Can be
'glob' or 'literal'. (default "glob")
--subgraph string           Path to Subgraph config file
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn relationship](#) - Perform Relationship related operations

# DDN CLI: ddn run

Run specific script from project's context config.

## Synopsis

Run custom scripts defined in project's context config (typically at `<project-root>/.hasura/context.yaml`)

```
ddn run <script-name> [flags]
```

## Examples

```
# Run script `docker-start` defined in project's context config
ddn run docker-start
```

## Options

```
-h, --help    help for run
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn subgraph

Perform Subgraph related operations.

## Synopsis

Perform Subgraph related operations

Alias: subgraphs

## Available operations

- [ddn subgraph add](#) - Add a Subgraph config file to Supergraph config file
- [ddn subgraph build](#) - Perform SubgraphBuild related operations
- [ddn subgraph init](#) - Initialize a new Subgraph in local metadata

## Options

```
-h, --help    help for subgraph
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn subgraph add

Add a Subgraph config file to Supergraph config file.

## Synopsis

Add a Subgraph config file to Supergraph config file

```
ddn subgraph add --subgraph <path-to-subgraph-config-file> --target-supergraph <path-to-supergraph-config-file> [flags]
```

## Examples

```
# Add a Subgraph config file "./app/subgraph.yaml" to the Supergraph config file "./supergraph.yaml"
ddn subgraph add --subgraph ./app/subgraph.yaml --target-supergraph ./supergraph.yaml
# Add a Subgraph config file "./app/subgraph.yaml" to multiple Supergraph config files"
ddn subgraph add --subgraph ./app/subgraph.yaml --target-supergraph ./supergraph.stg.yaml --target-supergraph ./supergraph.prod.yaml
```

## Options

-h, --help	help for add
--subgraph string file (required)	Path to Subgraph config
--target-supergraph stringArray	Supergraph config file to add the Subgraph. Can be repeated to provide multiple Supergraph config files (required)

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt          Do not prompt for required but missing  
flags  
--out string         Output format. Can be table, json or yaml.  
(default "table")  
--timeout int        Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn subgraph](#) - Perform Subgraph related operations

# DDN CLI: ddn subgraph build

Perform SubgraphBuild related operations.

## Synopsis

Perform SubgraphBuild related operations

## Available operations

- [ddn subgraph build apply](#) - Apply a Subgraph build on Hasura DDN
- [ddn subgraph build create](#) - Create a SubgraphBuild on Hasura DDN
- [ddn subgraph build get](#) - List SubgraphBuilds or get details of a specific one

## Options

```
-h, --help    help for build
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn subgraph](#) - Perform Subgraph related operations



# DDN CLI: ddn subgraph build apply

Apply a Subgraph build on Hasura DDN.

## Synopsis

Apply a Subgraph build on Hasura DDN

```
ddn subgraph build apply <subgraph-build-version> [flags]
```

## Examples

```
# Apply a Subgraph build to Project "pet-lion-2649"
ddn subgraph build apply <subgraph-build-version> --project
pet-lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string (default <current_context>)	Name of the context to use.
-h, --help	help for apply
-p, --project string	DDN Project name
--self-hosted-data-plane	Is the data plane self hosted?

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string (default "table")	Output format. Can be table, json or yaml.

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn subgraph build](#) - Perform SubgraphBuild related operations

# DDN CLI: ddn subgraph build create

Create a SubgraphBuild on Hasura DDN.

## Synopsis

Create a SubgraphBuild on Hasura DDN

```
ddn subgraph build create [flags]
```

## Examples

```
# Build the Subgraph from a config file for Project with some
environment variables
ddn subgraph build create --subgraph ./app/subgraph.yaml --
project pet-lion-2649 --env key1=val1
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-d, --description string	(Optional) description of the build
-e, --env stringArray	Environment variable, e.g. key=val. Can be repeated to provide multiple env vars
--env-file stringArray	Path to .env file. Can be repeated to provide multiple env files
-h, --help	help for create
--no-build-connectors	Do not recursively build all connectors in the subgraph and use their URLs for subgraph build. (default: false)
-p, --project string	DDN Project name
--self-hosted-data-plane	Is the data plane self hosted?

--subgraph string	Path to Subgraph config
file	
--target-env-file string	Env file to write the connector build URLs to.

--update-connector-link-schema    Update DataConnectorLink schema with the NDC schema of the connectors built recursively. (default: false)

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")
--timeout int	Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)

## Parent operation

- [ddn subgraph build](#) - Perform SubgraphBuild related operations

# DDN CLI: ddn subgraph build get

List SubgraphBuilds or get details of a specific one.

## Synopsis

List SubgraphBuilds or get details of a specific one

```
ddn subgraph build get [subgraph-build-version] [flags]
```

## Examples

```
# View details of a SubgraphBuild in the Project 'pet-lion-2649'  
ddn subgraph build get <subgraph-build-version> --project pet-lion-2649  
# List all SubgraphBuilds of a Project 'pet-lion-2649'  
ddn subgraph build get --project pet-lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for get
-p, --project string	DDN Project name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn subgraph build](#) - Perform SubgraphBuild related operations

# DDN CLI: ddn subgraph init

Initialize a new Subgraph in local metadata.

## Synopsis

Initialize a new Subgraph in local metadata

```
ddn subgraph init <subgraph-name> --dir <dir-name> [flags]
```

## Examples

```
# Initialize a Subgraph "app" in the directory "./app"
ddn subgraph init app --dir ./app
# Initialize a Subgraph "app" in the directory "./app" and add
it to Supergraph config files "./supergraph.yaml" and
"./supergraph.cloud.yaml"
ddn subgraph init app --dir ./app --target-supergraph
./supergraph.yaml --target-supergraph ./supergraph.cloud.yaml
```

## Options

--dir string	Directory to initialize the Subgraph (required)
-h, --help	help for init
--target-supergraph stringArray	Supergraph config file to add the Subgraph. Can be repeated to provide multiple Supergraph config files

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags

```
--out string          Output format. Can be table, json or yaml.  
(default "table")  
--timeout int         Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn subgraph](#) - Perform Subgraph related operations

# DDN CLI: ddn supergraph

Perform Supergraph related operations.

## Synopsis

Perform Supergraph related operations

Alias: supergraphs

## Available operations

- [ddn supergraph build](#) - Perform SupergraphBuild related operations
- [ddn supergraph init](#) - Initialize a new Supergraph project directory

## Options

```
-h, --help    help for supergraph
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn supergraph build

Perform SupergraphBuild related operations.

## Synopsis

Perform SupergraphBuild related operations

## Available operations

- [ddn supergraph build apply](#) - Apply a SupergraphBuild to its Project on Hasura DDN
- [ddn supergraph build create](#) - Create a SupergraphBuild on Hasura DDN
- [ddn supergraph build delete](#) - Delete a SupergraphBuild from a Project
- [ddn supergraph build diff](#) - See changes made to the GraphQL schema from one build version to another.
- [ddn supergraph build get](#) - List SupergraphBuilds or get details of a specific one
- [ddn supergraph build local](#) - Build the Supergraph and generate assets to run the local Engine
- [ddn supergraph build set-self-hosted-engine-url](#) - Set the engine's URL for a build for a project in a self hosted data plane.

## Options

```
-h, --help    help for build
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn supergraph](#) - Perform Supergraph related operations

# DDN CLI: ddn supergraph build apply

Apply a SupergraphBuild to its Project on Hasura DDN.

## Synopsis

Apply a SupergraphBuild to its Project on Hasura DDN

```
ddn supergraph build apply <supergraph-build-version> [flags]
```

## Examples

```
# Apply a SupergraphBuild to a Project "pet-lion-2649"
ddn supergraph build apply <supergraph-build-version> --
project pet-lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string (default <current_context>)	Name of the context to use.
-h, --help	help for apply
-p, --project string	DDN Project name
--self-hosted-data-plane	Is the data plane self hosted?

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string (default "table")	Output format. Can be table, json or yaml.

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn supergraph build](#) - Perform SupergraphBuild related operations

# DDN CLI: ddn supergraph build create

Create a SupergraphBuild on Hasura DDN.

## Synopsis

Create a SupergraphBuild on Hasura DDN

```
ddn supergraph build create [flags]
```

## Examples

```
# Build the Connectors and the Supergraph
ddn supergraph build create --supergraph supergraph.yaml --
project pet-lion-2649 --env-file .env.cloud
# Build the Supergraph without building the Connectors
ddn supergraph build create --supergraph supergraph.yaml --
project pet-lion-2649 --no-build-connectors
# Build the Supergraph and update the link schema and target
env file
ddn supergraph build create --supergraph supergraph.yaml --
project pet-lion-2649 --update-connector-link-schema --target-
env-file .env.cloud
# Build a composed Supergraph using Subgraphs with build
versions and using the applied Supergraph Build as the base
Supergraph (Advanced plan only)
ddn supergraph build create --subgraph-version
globals:c15b0b4031 --subgraph-version my_subgraph:c15b0b4031 --
base-supergraph-on-applied
# Build a composed Supergraph using Subgraphs with build
versions and base Supergraph version (Advanced plan only)
ddn supergraph build create --subgraph-version
globals:c15b0b4031 --subgraph-version my_subgraph:c15b0b4031 --
base-supergraph-version c15b0b4871
```

## Options

--base-supergraph-on-applied	Use the applied Supergraph as the base supergraph
--base-supergraph-version string	Base Supergraph version for the compose build
--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-d, --description string	(Optional) description of the build
-e, --env stringArray	Environment variable, e.g. key=val. Can be repeated to provide multiple env vars
--env-file stringArray	Path to .env file. Can be repeated to provide multiple env files
-h, --help	help for create
--no-build-connectors	Do not recursively build all connectors in all subgraphs and use their URLs for supergraph build. (default: false)
--no-diff	Do not do a schema diff against the applied build
--output-dir string	Path to the directory to output the build artifacts
-p, --project string	DDN Project name
--self-hosted-data-plane	Is the data plane self hosted?
--subgraph-version stringArray	Subgraph(s) with build version to compose
--supergraph string	Path to Supergraph config file
--target-env-file string	Env file to write the connector build URLs to.
--update-connector-link-schema	Update DataConnectorLink schema with the NDC schema of the connectors built recursively. (default: false)

## Options inherited from parent operations

--log-level string Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")

```
--no-prompt          Do not prompt for required but missing
flags
--out string         Output format. Can be table, json or yaml.
(default "table")
--timeout int        Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn supergraph build](#) - Perform SupergraphBuild related operations

# DDN CLI: ddn supergraph build delete

Delete a SupergraphBuild from a Project.

## Synopsis

Delete a SupergraphBuild from a Project

```
ddn supergraph build delete <supergraph-build-version> [flags]
```

## Examples

```
# Delete a SupergraphBuild from a Project "pet-lion-2649"
ddn supergraph build delete <supergraph-build-version> --  
project pet-lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for delete
-p, --project string	DDN Project name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")
--timeout int	Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)

## Parent operation

- [ddn supergraph build](#) - Perform SupergraphBuild related operations

# DDN CLI: ddn supergraph build diff

See changes made to the GraphQL schema from one build version to another..

## Synopsis

See changes made to the GraphQL schema from one build version to another.

```
ddn supergraph build diff <build-version-1> <build-version-2>  
[flags]
```

## Examples

```
# Compare changes made to the GraphQL schema from build version  
"qfrr5e5jyw" to "g6v6nh73h0" for Project "pet-lion-2649"  
ddn supergraph build diff qfrr5e5jyw g6v6nh73h0 --project pet-  
lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for diff
-p, --project string	DDN Project name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn supergraph build](#) - Perform SupergraphBuild related operations

# DDN CLI: ddn supergraph build get

List SupergraphBuilds or get details of a specific one.

## Synopsis

List SupergraphBuilds or get details of a specific one

```
ddn supergraph build get [supergraph-build-version] [flags]
```

## Examples

```
# View details of a SupergraphBuild in the Project "pet-lion-2649"  
ddn supergraph build get <supergraph-build-version> --project pet-lion-2649  
# List all SupergraphBuilds of a Project "pet-lion-2649"  
ddn supergraph build get --project pet-lion-2649
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-h, --help	help for get
-p, --project string	DDN Project name

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing flags
--out string	Output format. Can be table, json or yaml. (default "table")

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn supergraph build](#) - Perform SupergraphBuild related operations

# DDN CLI: ddn version

Prints the CLI version.

## Synopsis

Use this command to print the current version of the CLI. This command can also be used to check if a new version of the CLI is available.

```
ddn version [flags]
```

## Options

```
-h, --help    help for version
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing flags
--out string          Output format. Can be table, json or yaml. (default "table")
--timeout int         Request timeout in seconds [env: HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface



# DDN CLI: ddn supergraph build local

Build the Supergraph and generate assets to run the local Engine.

## Synopsis

Build the Supergraph and generate assets to run the local Engine using the DDN hosted metadata build service.

Any relationships to subgraphs defined in other repositories will not be validated and will be ignored.

```
ddn supergraph build local [flags]
```

## Examples

```
# Build the Supergraph using the supergraph config and local
env file from context and output it to default engine directory
i.e. <project-root>/engine/build
ddn supergraph build local
# Build the Supergraph using a specific supergraph config file
and env file and output it to a specific directory
ddn supergraph build local --output-dir <path-to-engine-
directory> --supergraph supergraph.yaml --env-file .env
```

## Options

--ci	Disables the use of context
-c, --context string	Name of the context to use. (default <current_context>)
-e, --env stringArray	Environment variable, e.g. key=val. Can be repeated to provide multiple env vars
--env-file stringArray	Path to .env file. Can be repeated to provide multiple env files
-h, --help	help for local
--output-dir string	Path to the engine directory to

```
output the build artifacts. (defaults to `<project-root>/engine/build`)
--supergraph string      Path to Supergraph config file
```

## Options inherited from parent operations

```
--log-level string    Log level. Can be DEBUG, WARN, INFO,
ERROR, or FATAL. (default "INFO")
--no-prompt           Do not prompt for required but missing
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn supergraph build](#) - Perform SupergraphBuild related operations

# DDN CLI: ddn supergraph build set-self-hosted-engine-url

Set the engine's URL for a build for a project in a self hosted data plane..

## Synopsis

Set the engine's URL for a build for a project in a self hosted data plane.

```
ddn supergraph build set-self-hosted-engine-url <url> --build-version <build-version> [flags]
```

## Examples

```
# Set build URL to "example.com:3000" for project "pet-lion-2649"
ddn supergraph build set-self-hosted-engine-url
example.com:3000 --build-version <build-version> --project pet-lion-2649
```

## Options

--build-version string	SupergraphBuild version (required)
--ci	Disables the use of context
-c, --context string (default <current_context>)	Name of the context to use.
-h, --help	help for set-self-hosted-engine-url
-p, --project string	DDN Project name

## Options inherited from parent operations

```
--log-level string Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
```

```
--no-prompt          Do not prompt for required but missing
flags
--out string         Output format. Can be table, json or yaml.
(default "table")
--timeout int        Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn supergraph build](#) - Perform SupergraphBuild related operations

# DDN CLI: ddn supergraph init

Initialize a new Supergraph project directory.

## Synopsis

Initialize a new Supergraph project directory

```
ddn supergraph init <path-to-project-dir> [flags]
```

## Examples

```
# Initialize a new Supergraph project directory with a default
# subgraph 'app'
ddn supergraph init <path-to-project-dir>
# Initialize a new Supergraph project directory with a subgraph
# 'mysg'
ddn supergraph init <path-to-project-dir> --create-subgraph
mysg
```

## Options

--create-subgraph string	The default subgraph to add (default "app")
--globals-subgraph string	Name of the globals subgraph (default "globals")
-h, --help	help for init
--no-globals-subgraph	Do not add a globals subgraph
--no-subgraph	Do not add a default subgraph

## Options inherited from parent operations

--log-level string	Log level. Can be DEBUG, WARN, INFO, ERROR, or FATAL. (default "INFO")
--no-prompt	Do not prompt for required but missing

```
flags
--out string          Output format. Can be table, json or yaml.
(default "table")
--timeout int         Request timeout in seconds [env:
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn supergraph](#) - Perform Supergraph related operations

# DDN CLI: ddn update-cli

Update this CLI to the latest version or to a specific version.

## Synopsis

You can use this command to update the CLI to the latest version or a specific version.

```
ddn update-cli [flags]
```

## Examples

```
# Update CLI to latest version:  
ddn update-cli  
# Update CLI to a specific version (say v1.0.0):  
ddn update-cli --version v1.0.0  
# To disable the auto-update check on the CLI, set  
# "show_update_notification": false  
# in ~/.ddn/config.yaml
```

## Options

```
-h, --help            help for update-cli  
--version string    A specific version to install
```

## Options inherited from parent operations

```
--log-level string  Log level. Can be DEBUG, WARN, INFO,  
ERROR, or FATAL. (default "INFO")  
--no-prompt         Do not prompt for required but missing  
flags  
--out string        Output format. Can be table, json or yaml.  
(default "table")
```

```
--timeout int      Request timeout in seconds [env:  
HASURA_DDN_TIMEOUT] (default 100)
```

## Parent operation

- [ddn](#) - DDN Command Line Interface