In simple terms, an environment variable in the context of dbt (data build tool) is a configuration setting that you can define outside of your dbt project. It is a value that can be accessed by the dbt tool and its associated processes, providing a way to customize and influence the behavior of dbt without modifying your project's code directly.

Environment variables are global variables that can be set at the operating system level or within your development environment. They are referenced by name and hold values that can be used by various programs and tools, including dbt.

In the case of dbt, specific environment variables are used to control certain aspects of dbt's behavior, such as the location of configuration files, database connections, and other settings. By setting these variables, you can customize how dbt behaves without modifying your dbt project code.

he purpose and use of environment variables in dbt can be summarized as follows:

1. Configuration Management: Environment variables serve as a flexible and centralized way to manage configuration settings for dbt. Instead of hardcoding configuration values in your dbt project code, you can use environment variables to set and modify these values externally.
2. Portability and Scalability: By using environment variables, you can make your dbt project more portable and scalable. Environment variables allow you to easily move your project across different environments (e.g., development, staging, production) without modifying the code. This simplifies deployment and makes it easier to maintain consistent configurations across different environments.
3. Security and Secrets Management: Environment variables can be used to store sensitive information, such as database credentials or API keys, outside of your codebase. This helps improve security by separating sensitive information from the project code and reduces the risk of accidental exposure.
4. Flexibility and Adaptability: Environment variables provide flexibility to adapt your dbt project to different setups and requirements. You can modify the behavior of dbt by adjusting the values of environment variables without making code changes. This makes it easier to handle variations in different environments or accommodate specific deployment scenarios.
5. Collaboration and Teamwork: Environment variables allow teams to collaborate on a dbt project more effectively. By using a shared set of environment variables, team members can synchronize their development environments and ensure consistent configurations across different machines.
6. Continuous Integration and Deployment (CI/CD): Environment variables play a crucial role in CI/CD workflows. They allow you to configure and control dbt's behavior during automated build, test, and deployment processes. Environment variables can be set and managed within CI/CD tools to ensure consistent and reproducible builds.

In summary, environment variables in dbt provide a way to manage configuration settings, improve portability, enhance security, offer flexibility, support collaboration, and enable seamless integration with CI/CD workflows. They are a powerful tool for customizing and adapting dbt projects to different environments and requirements.

About env\_var function

The env\_var function can be used to incorporate Environment Variables from the system into your dbt project. This env\_var function can be used in your **profiles.yml file**, the **dbt\_project.yml file,** the **sources.yml file,** your **schema.yml files,** and in **model .sql files.**

*Essentially env\_var is available anywhere dbt processes jinja code.*

When used in a profiles.yml file (to avoid putting credentials on a server), it can be used like this:

**profiles.yml**

profile:  
 target: prod  
 outputs:  
 prod:  
 type: postgres  
 host: 127.0.0.1  
 *# IMPORTANT: Make sure to quote the entire Jinja string here*  
 user: **"{{ env\_var('DBT\_USER') }}"**  
 password: **"{{ env\_var('DBT\_PASSWORD') }}"**

If the **DBT\_USER** and **DBT\_PASSWORD** environment variables are present when dbt is invoked, then these variables will be pulled into the profile as expected. If any environment variables are not set, then dbt will raise a **compilation error.** So to avoid these error you should define the default variable so that In case if env\_var is not defined in environments then the default values can be set

          database: '{{env\_var("DBT\_CHICAGOCRIME\_DATABASE",**"demo"**)}}'

INTEGER ENVIRONMENT VARIABLES

*If passing an environment variable for a property that uses an* ***integer type*** *(for example, port, threads), be sure to add a filter to the Jinja expression, as shown here. Otherwise, dbt will raise an****['threads']: '1' is not of type 'integer' error. {{ env\_var('DBT\_THREADS') | int }}****or****{{ env\_var('DB\_PORT') | as\_number }}***

QUOTING, CURLY BRACKETS, & YOU

*Be sure to quote the entire jinja string (as shown above), or else the YAML parser will be confused by the Jinja curly brackets.*

env\_var accepts a second, optional argument for default value, like so:

          database: '{{env\_var("DBT\_CHICAGOCRIME\_DATABASE",**"demo"**)}}'

Secrets

For certain configurations, you can use "secret" env vars. Any env var named with the prefix **DBT\_ENV\_SECRET\_** will be:

* Available for use in **profiles.yml + packages.yml,** via the same **env\_var()** function
* Disallowed everywhere else, including **dbt\_project.yml** **and model SQL**, to prevent accidentally writing these secret values to the data warehouse or metadata artifacts
* Scrubbed from dbt logs and replaced with \*\*\*\*\*, any time its value appears in those logs (even if the env var was not called directly)

*The primary use case of secret env vars is git access tokens for*[*private packages*](https://docs.getdbt.com/docs/build/packages#private-packages)*.*

* *When dbt is loading profile credentials and package configuration, secret env vars will be replaced with the string value of the environment variable. You cannot modify secrets using Jinja filters, including type-casting filters such as [as\_number](https://docs.getdbt.com/reference/dbt-jinja-functions/as_number) or [as\_bool](https://docs.getdbt.com/reference/dbt-jinja-functions/as_bool), or pass them as arguments into other Jinja macros. You can only use one secret per configuration:*

*#* ***works***  
host: "{{ env\_var('DBT\_ENV\_SECRET\_HOST') }}"  
  
***# does not work***  
host: "www.{{ env\_var('DBT\_ENV\_SECRET\_HOST\_DOMAIN') }}.com/{{env\_var('DBT\_ENV\_SECRET\_HOST\_PATH') }}"

**Environment variables**

Environment variables can be used to customize the behavior of a dbt project depending on where the project is running. See the docs on [env\_var](https://docs.getdbt.com/reference/dbt-jinja-functions/env_var) for more information on how to call the jinja function {{env\_var('DBT\_KEY','OPTIONAL\_DEFAULT')}} in your project code.

**ENVIRONMENT VARIABLE NAMING AND PREFIXING**

Environment variables in dbt Cloud must be prefixed with either DBT\_ or DBT\_ENV\_SECRET\_. Environment variables keys are uppercased and case sensitive. When referencing {{env\_var('DBT\_KEY')}} in your project's code, the key must match exactly the variable defined in dbt Cloud's UI

Environment variables

Job Level > Environment Level > Project Level > Default Argument

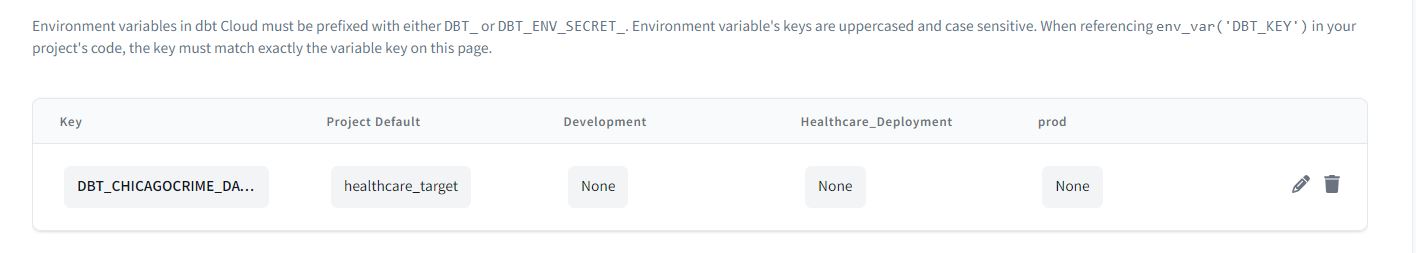
1. Default Argument:

  database: '{{env\_var("DBT\_CHICAGOCRIME\_DATABASE",**"demo"**)}}'

Default argument Is used when env\_var is defined. This is because we will have to pass a certain value in "DBT\_CHICAGOCRIME\_DATABASE"

From anywhere but if it is not passed then it could raise a compilation error, so inorder to avoid that error we use default value so it replaces with its value the empty space when no value is passed.

It can also be defined from environment variables

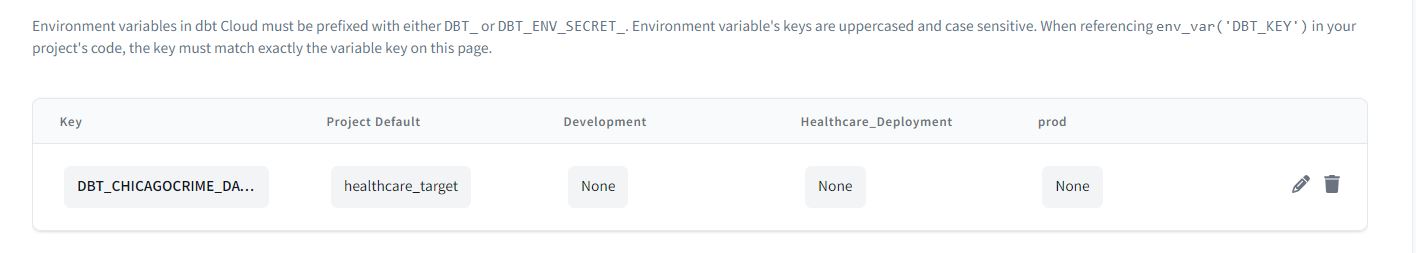


If no database value is inserted it will use **“demo”** database

**It has the lowest precedence** because if the variable is passed with value from higher precedence it will over ride the default value. Now **“demo”** is over ridden by **“healthcare\_target”**

1. Project Level:

We have to pass the project level variable value from the environments. If it not defined then the default value **“demo”** will be considered.



We have passed the **“healthcare\_target”** as a project level value so it will over ride the default and be considered by the **dbt\_project.yml**

**APPROPRIATE COVERAGE**

*If you have not set a project level default value for every environment variable, it may be possible that dbt Cloud does not know how to interpret the value of an environment variable in all contexts. In such cases, dbt will throw a compilation error: "Env var required but not provided".*

**CHANGING ENVIRONMENT VARIABLES MID-SESSION IN THE IDE**

*If you change the value of an environment variable mid-session while using the IDE, you may have to refresh the IDE for the change to take effect.*

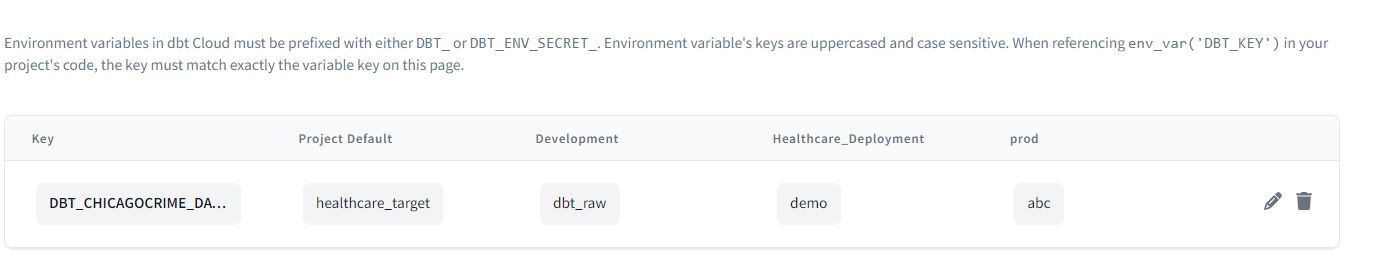
1. Environment level:

Now we have to define some value for the whole environment.

We have 3 env defined that Is

1. Development
2. Healthcare\_deployment
3. prod

so when we pass the value inside it for the env\_var function it will over ride project level variable and we can use separate values / databases for separate environments.



Now if I am working on development environment it will use the value = ‘dbt\_raw’ and put it inside the env\_var. this is actually the database name but we can use it for any purpose.

If I am working on env it will use demo as the value and will pass it and if prod env then abc will be used.