**Environments in dbt**

In software engineering, environments are used to enable engineers to develop and test code without impacting the users of their software.

“Production” (or prod) refers to the environment that end users interact with, while “development” (or dev) is the environment that engineers work in. This means that engineers can work iteratively when writing and testing new code in development, and once they are confident in these changes, deploy their code to production.

In traditional software engineering, different environments often use completely separate architecture. For example, the dev and prod versions of a website may use different servers and databases.

Data warehouses can also be designed to have separate environments – the \_production\_ environment refers to the relations (for example, schemas, tables, and views) that your end users query (often through a BI tool).

**Environments, jobs, and runs**

Environments encompass a collection of settings for how you want to run your dbt project. This includes:

* dbt version
* git branch
* data location (target schema)

1. The **development environment** applies the same settings for all developers working in dbt Cloud as they work in team. Deployment environments can be set up to support various different deployment strategies and architectures.
2. **Jobs** are a set of dbt commands that you run within an environment. This can include commands like dbt build with any selection syntax / flags that you may want. These jobs can then be kicked off through a variety of means.
3. **Runs** are the implementation of a specific job that you have configured that was triggered. While a job is running, you can see that status of that job in real time. Once the job is finished, you can see the run results and view artifacts from that particular run.

**Implementation Suggestion**

We recommend the 1 trunk / direct promotion approach for setting up your environment.  Here are some quick steps to guide you.

Check that your development environment does **not** have a custom branch enabled (consider upgrading your dbt version while you are at it too!)

Create a new deployment environment called **Production** or**Deployment**with the same version of your development environment for compatibility (we recommend the latest though

**dbt build**

**dbt build** is a smart combination of **dbt run**and **dbt** **test** that will also build your snapshots and seeds.  All of this is done in DAG order from left to right.  This is particularly helpful for **only continuing**to build your DAG downstream from a node if the tests pass on that node.

**Common deployment jobs**

We covered four common deployment jobs:

**Standard job -**this jobs is typically just **dbt build** and will rebuild your entire DAG while **including**incremental logic.

**Full refresh job -**this job similar will use **dbt build** except it will pass the full refresh flag. This will force incremental models to be dropped and materialized from scratch and seeds to be dropped and rebuilt as well.

**Time sensitive**- this is a job that usually has a time sensitive business use case. You might consider refreshing marketing data or sales data more frequently than your standard job. You can accomplish this by using model selector syntax with your commands.

**Fresh rebuild** - fresh rebuild allows you to **only** rebuild models downstream from sources that have been updated since the previous run.

**One off vs. unified jobs**

There are chances that you have jobs that run at the same time and rebuild some share resources. In these cases, it can be helpful to use model selector syntax that allows you to select the intersection or union of models rather than running jobs individually.

<https://docs.getdbt.com/docs/deploy/cloud-ci-job#fresh-rebuilds>

### Coordinating different jobs

There are cases where you may have jobs that run concurrently on shared models. In these cases, you likely want to account for those conflicts. This can be difficult to achieve with the user friendly scheduling options. To avoid these conflict, consider using cron syntax or API calls to avoid the overlapping jobs.