# **Installations**

This guide will take you through the process of installing Git and Docker, as well as running a container using Docker Compose from a Git repository. This guide is only relevant for windows.

## **Prerequisites**

Before you begin, ensure that you have the following:

1. A terminal or command prompt on your computer.
2. Administrative privileges on your computer.
3. A working internet connection.
4. Visual Studio Code.

## **Instructions**

### **Step 1: Install Git**

Git is a version control system that is used to manage source code. To install Git on your computer, follow the steps below:

1. Open your terminal or command prompt.
2. On Windows, download the Git installer from <https://git-scm.com/downloads> and follow the prompts to install Git. On Mac and Linux, Git is typically pre-installed.
3. Once Git is installed, open your terminal or command prompt and run the following command to verify that Git is installed: git --version
4. If Git is installed correctly, you should see the version number displayed in your terminal or command prompt.

### **Step 2: Install Docker**

Docker is a platform that allows you to create, deploy, and run applications in containers. To install Docker on your computer, follow the steps below:

1. Open your terminal or command prompt.
2. Download the Docker Desktop installer from <https://docs.docker.com/desktop/install/windows-install/>   
   and follow the prompts to install Docker Desktop.
3. Once Docker is installed, open your terminal or command prompt and run the following command to verify that Docker is installed: docker --version
4. If Docker is installed correctly, you should see the version number displayed in your terminal or command prompt.

### **Step 3: Clone Git Repository**

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to clone the Git repository.
3. Run the following command to clone the repository:   
   git clone https://github.com/noamelli/carasso.git

### **Step 4: Run Airflow Using Docker Compose**

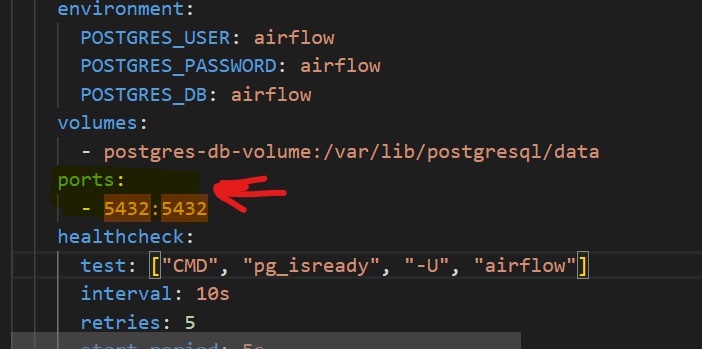
1. Open your terminal or command prompt.
2. Navigate to the directory where you cloned the Git repository.
3. Make sure Docker compose was successfully installed by this command: docker-compose – version
4. Run these commands in order to create a project folder in visual studio code:
   1. mkdir airflow-docker
   2. cd airflow-docker
   3. code .

* It will open the VS code - open a new terminal.

1. Go to the part of fetching docker-compose.yaml in Airflow documentation and copy the path without this: -Lf0. At the end of the path add this: ‘docker-compose.yaml’. After pasting in the terminal, you will get the yaml file, which is used to define and configure Docker applications.
2. Run the following commands:
   1. mkdir dags
   2. mkdir logs
   3. mkdir plugins
3. Run the following command to initialize airflow: docker compose up airflow-init
4. Run the following command to start the containers defined in your Docker Compose file: docker compose up
5. Make sure the docker desktop is open, unless you won't be able to enter the airflow web server.
6. Open localhost:8080 and sign in with username and password (the default value is airflow)
7. If you need to close all the docker containers use this command: docker compose down

### **Step 5: Connect Airflow to Postgresql**

1. Open the yaml file in VS code and add these lines:



1. Command this line in the terminal: docker-compose up -d --no-deps --build postgres.
2. Download DBeaver Community for windows.Its a free cross-platform database tool. Our tables will be stored in this DB.
3. Open the tool, connect to a database and choose postgresql.
4. fulfill the details:
   1. connect by: host.
   2. host: localhost.
   3. database: postgres.
   4. port: 5432.
   5. username and password of the airflow web server.
5. In the postgresql tab choose show all databases.
6. Press test connection. It might ask us to install some drivers. We will make sure the connection ends successfully.
7. Create a new database in dbeaver.
8. create a new connection to postgres : airflow web server → admin → connection → new connection.
   1. conn id: postgres\_localhost.
   2. conn type: postgres.
   3. database : the name of the db we created.
   4. login and password of airflow web server.
   5. port :5432.
   6. host: host.docker.internal.

# **ETL Process Description**

1. Create all of the tables if they do not exist.
2. Truncate MRR and STG tables, so we can load them with new records.
3. Backup DWH tables - truncate backup tables and then load them with the records in DWH tables.
4. Extract data from Excel and load to MRR tables. Unlike the other tasks, this one uses a python operator and executes functions that use pandas in order to read from Excel.
   1. Dim tables - the initial loading to MRR dim tables is full.
   2. Fact tables - the initial loading to MRR fact tables is incremental. We will load new orders that are not shown in the DWH tables.   
      There are two MRR fact tables: orders and details. The details table shows the quantity sold for each product in each order.
      1. The reason for this is the idea that an order that happened in the past will not change, so the record will not be updated either.
      2. Therefore, in the following steps, the loading to STG and DWH fact tables will be full.
      3. We will locate orders that are shown in the DWH tables based on the order Id key.
   3. The data type of all columns in MRR tables is varchar, in case the data shown in the file doesn't match the desired format. This is how we avoid failure of the program.
5. Loading data from MRR Dim tables to STG Dim tables.
   1. The Loading is full.
   2. We will cast the column data type to the desired one.
   3. We will check whether the format of the value inserted is correct. If it's wrong or the value is null - we will change the format or insert the agreeable one (0, null, etc..).
   4. Generating dates and time tables.
6. Loading data from STG Dim tables to DWH Dim tables.
   1. Truncate DWH Dates and time tables, so we can load them with the relevant records (full load).
   2. We will treat customers, employees and products as slowly changing dimension type 2 and save versions when one of the next columns value change:
      1. Customers - city, country.
      2. Products - unit\_cost, unit\_price, supplier.
      3. Employees - reports\_to, title, month\_salary, office.
   3. The loading consists of three stages:
      1. Inserting all of the new records based on the primary key of the STG table. When a record is being added, it automatically gets DW value - a column identifier that gets the value: identity (1,1). It also gets a valid from value (default - current date) and the valid until will be null.
      2. Updating the valid until in old records that at least one of the relevant columns has changed.
      3. Inserting new records with the new value. The valid from value is the current date and the valid until is null.
7. Loading data from MRR fact tables to STG fact tables.
   1. The loading is full because we already loaded incremental records to MRR.
   2. We will cast the column data type to the desired one.
   3. We will check whether the format of the value inserted is correct. If it's wrong or the value is null - we will change the format or insert the agreeable one (0, null, etc..).
8. Referential Integrity - if the fact tables contain dimensions that are not shown in DWH Dim tables we will mark their key and set it to 0.
9. Loading data from STG fact tables to DWH fact table -Product\_In\_Order.
   1. The loading is full.
   2. We will combine STG\_Fact\_Orders with STG\_Fact\_Details using inner join. We will also combine DWH Dim tables and select only the most recent records (where valid until = null).
   3. we will create calculated fields : total\_price, total\_cost.
10. Updating DWH\_Snapshot\_Customers\_Transactions. This table shows the transaction numbers (new customers, abandoned, etc..) in each year, quarter and month.

**Snapshot table logic**

1. Updating current\_dates table - a one record view showing:
   * Start date of previous month.
   * last date of previous month.
   * Previous month.
   * Year of previous month.
   * Quarter of previous month.
2. Loading Snapshot\_Customers\_Transactions\_month - a view that shows for each customer the start date of the previous month and his status.
   * The first date of the month will be the Start date from the current\_dates table, which means the table will only contain a reference to the previous month and will be updated regularly.
   * The use of the month start date field stems from the idea that the view is monthly and not daily, but on the other hand we want to store the month and the year.
   * For each customer we will check his status in the previous month according to the calculation mentioned next.
3. Next we will check whether the Snapshot\_Customers\_Transactions\_Arch table contains the start of the previous month. If so, it means we have already loaded the previous month statuses, so we will stop at this point. Otherwise, we will continue to the next step.
4. Loading Snapshot\_Customers\_Transactions\_Arch - a table containing historical data. It shows for each customer his status at each start date of the month.
5. Loading DWH\_Snapshot\_Customers\_Transactions.
   1. Incremental loading - we will load only breakdowns combinations that are not shown in the table (year, month, quarter, county, city).
   2. We will use Inner join with DWH\_Dim\_Customers, in order to show the countries and the cities.

**Logic of the status calculation**

1. New - a customer that the year and month of the minimum valid from shown in DWH\_Dim\_Customers match the previous month.   
   The reason for selecting the minimum is that a customer might change his address and then the new record will get an older valid until.
2. Abandoned - a customer who hasn't bought for the last 3 months (including the month we refer to).
3. Reactivated - a customer who was marked as an abandoner two months ago and in the previous month made a purchase.
4. regular - a customer who isn't new or did not abandon/Reactivate.

**Business assumptions:**

1. Every time we update the snapshot table, we refer to the months that have ended. This is because we don't want to measure customer behavior in a month that hasn't ended yet. For example, it is possible that a customer did not have time to purchase this month and will purchase later, so we would not want to categorize him as “Abandoned “.
2. Following on from the previous section, since we are always looking at months that have ended, there is no point in updating the snapshot table every run during the month, because there will be no difference in the data. We do not refer to the current month, so the snapshot table will be updated once a month.