DBMQ

Release 2.1

Ali Reza Yahyapour

Feb 19, 2021

TABLE OF CONTENTS:

1 Why DBMQ is Needed **3**
- 1.1 `webserver` global configurations 3

2 Installation **5**
- 2.1 Docker Installation 5
- 2.2 DBMQ Installation 5

Docker-based Message Queuing (DBMQ) is an efficient way to run the pre-built configurations on the build process of Dockerfiles. Once you have finished configuring, you will be able to create your images based on your configurations.

Warning: DBMQ is fully stable on Linux-based distributions and there is no guarantee that this tool works as properly as what it does on the Linux machines on Windows machines.

CHAPTER

ONE

WHY DBMQ IS NEEDED

This system locates between the user and the Docker service. You config your requirements and let this automated system to provide them to you. You don't need to be a genius in Docker, just keep configuring.

1.1 `webserver` global configurations

1.1.1 Introduction

All essential configurations are stored in a file named `webserver.py` in the root directory of the project. Make sure you have already switched into the root directory of the project. Use a text editor in order to make changes.

You configure your images, containers, tags, environment variables, and etc in your `webserver.py` file. These configurations you write in the file will be authorized in the running process. The authorizer is an instance from the `BaseModel` class in the pydantic module.

1.1.2 `SERVER_CONFIGS` configurations

Server configurations take three parameter in a variable name `SERVER_CONFIGS` in the `webserver.py` file. This variable has to be filled with the proper values. It's kind of an essential argument to the main process.

CONTAINER It refers to the container that contains the core distribution. Mostly it's `settings.Image.centos_8` as `IMAGE` by default. `NAME` parameter is equal to tag in the Docker concepts. Use this parameter in order to pick a name for your core container.

- `IMAGE` : The image you want to serve your core container on.
- `NAME` : Tag name you want to name your container to.
- `NOCACHE` : It allows you to specify whether you want to use caches or not. It's set to `False` by default which means it does use the caches.

```
SERVER_CONFIGS = {
    ...
    'CONTAINER': {
        'IMAGE': settings.Image.centos_8,
        'NAME': 'django_core',
    },
    ...
}
```

NAME You can specify your Django core project name with the `NAME` parameter. It only inputs a name as follows.

```
SERVER_CONFIGS = {
    ...
    'NAME': 'sample',
    ...
}
```

This parameter will be executed in the following way so make sure you will never need to change this name.

```
$ django-admin startproject sample .
```

SERVER This parameter refers to the server you are building in your containers. Only `settings.Server.django` is available by now. It's like the webserver you are going to setup within your Docker containers.

```
SERVER_CONFIGS = {
    ...
    'SERVER': settings.Server.django,
    ...
}
```

Warning: Make sure the keys in the `SERVER_CONFIGS` are all uppercase. All validators are case-sensitive. For example, we have the `SERVER_CONFIGS.CONTAINER` validator right below.

```
class Container(BaseModel):
    IMAGE: str
    NAME: str
    NOCACHE: bool = False
```

CHAPTER

TWO

INSTALLATION

This installation process is valid on the Linux-based distributions so that you better to find the equivalent commands on other machines. The things you need to do is to install `Docker` engine on your machine and make sure you have `python>=3.8` available on your machine. We will catch all steps one by one. Let's have a general view on the way that we want to get through. First of all, we will install Docker and then, we will take a closer look on python and `virtualenvs`.

Note: We are going to install all dependencies through an isolated environment called `env`.

2.1 Docker Installation

Depending on your distribution, you need to find the way that you can install Docker on your machine. Here is the Official Docker Installation Guide that will help you to setup Docker on your distribution. The focuses are more on DBMQ and all its dependencies.

2.2 DBMQ Installation

In this section, you need to clone the project on your local machine and start installing the dependencies. Use the following command in order to clone the repository.

```
$ git clone https://github.com/dbmqproject/dbmq.git/
```

Once it's done, everything would be ready for the virtual environment. We are keeping up with `virtualenv` package which is also available on PyPi. In this example, we are using `python3.8` with `pip3`. Make sure you have already installed them on your machine, then run the following command to install the `virtualenv` on your machine.

```
$ pip3 install virtualenv
```

Once it's done, make sure you are already in the cloned DBMQ repository on your machine with the following command and create a new virtualenv named `env` within the directory. We have also activate our environment using the `source` command as follows.

```
$ pwd
/path/to/dir/DBMQ/
$ virtualenv env && source env/bin/activate
```

Note: Some developers wish to name their virtualenv `.venv` or `.env`. It actually makes the directory hidden which is much prettier.

Now, you are entered through your isolated environment. It's time to install the dependencies. DBMQ stores all dependencies in a text file named `requirements.txt`. Run the following command and it installs all dependencies automatically.

```
(env)$ pip install -r requirements.txt
```

Congratulations. Everything is ready to use. DBMQ has a default configuration that allows you to setup your applications as an example. It's time to crack on with the next step which is configuring our services.