Using .env Files for Environment Variables in Python Applications

Jake Witcher 24 dic 2020

Applications are made to be deployed. At some point during development you will need to think about the environment in which your application will run and the potentially sensitive or environment specific information your application will need to perform its tasks.

Environment variables are one of the key ways software developers provide an application with this kind of information, however running and testing an application locally that is dependent on environment variables can be a hassle if you are setting those variables on your local machine's environment.

The process of setting or changing an environment variable is time consuming and over time the number of environment variables you have to manage grows out of control. Eventually naming conflicts becomes an issue and every new variable requires a lengthy prefix to distinguish itself from similar variables.

Using a `.env` file will enable you to use environment variables for local development without polluting the global environment namespace. It will also keep your environment variable names and values isolated to the same project that utilizes them.

A `.env` file is a text file containing key value pairs of all the environment variables required by your application. This file is included with your project locally but not saved to source control so that you aren't putting potentially sensitive information at risk.

```
# environment variables defined inside a .env file
GCP_PROJECT_ID=my-project-id
SERVICE_ACCOUNT_FILE=path/to/serviceAccountCredentials
STORAGE_BUCKET_NAME=my-super-important-data
```

Nearly every programming language has a package or library that can be used to read environment variables from the `.env` file instead of from your local environment. For Python, that library is *python-dotenv*. Once the library is installed, an average use case for *python-dotenv* only requires adding two lines of code to your project.

```
from dotenv import load_dotenv

load_dotenv()
```

`load_dotenv()` will first look for a `.env` file and if it finds one, it will load the environment variables from the file and make them accessible to your project like any other environment variable would be.

```
import os
from dotenv import load_dotenv
```

```
load_dotenv()

GCP_PROJECT_ID = os.getenv('GCP_PROJECT_ID')
SERVICE_ACCOUNT_FILE = os.getenv('SERVICE_ACCOUNT_FILE')
STORAGE_BUCKET_NAME = os.getenv('STORAGE_BUCKET_NAME')
```

If an environment variable is not found in the `.env` file, `load_dotenv` will then search for a variable by the given name in the host environment. This means that when your project is running locally and the `.env` file is present, the variables defined in the file will be used. When your project is deployed to a host environment like a virtual machine or Docker container where the `.env` file is not present, the environment variables defined in the host environment will be used instead.

By default `load_dotenv` will look for the `.env` file in the current working directory or any parent directories however you can also specify the path if your particular use case requires it be stored elsewhere.

```
from dotenv import load_dotenv
from pathlib import Path

dotenv_path = Path('path/to/.env')
load_dotenv(dotenv_path=dotenv_path)
```

For most applications, that's all the information you should need to be productive with *python-dotenv* however there are a few additional features which you can read about in the *python-dotenv* documentation.

One of the benefits of using a `.env` file is that it becomes much easier to develop with environment variables in mind early on. Thinking about these things at an earlier stage of the development process will make it easier

for you to get your application ready for deployment.

If you are deploying your application in a Docker container, you can seamlessly transition to running and testing your application in a locally run container by using the flag `--env-file .env` with your `docker run` command. Docker will then set the same environment variables you've been using through *python-dotenv* in the container's environment.

Without a `.env` file, environment variables are a cumbersome albeit necessary part of testing your application locally. By using a `.env` file and the *python-dotenv* library, working with environment variables becomes much more manageable and will get you developing with deployment in mind right from the start.