



# Separate Configuration from Code

---

# Configuration in Code

Programmers sometimes "hard code" configuration data in code. Using the example below:

1. What '*configuration data*' is being stored in code?
2. Why is this a bad idea?

```
def connect_to_database():  
    """Open a connection to the database.  
       Returns: connection to database.  
    """  
    conn = MySQLdb.connect('pollsdb',  
                           user='polls',  
                           password='stupid')  
    return conn
```

Python has a standard DB-API that supports most databases.

# Problems with Configuration in Code

---

1. Effort to modify when configuration must change
  - and you may make mistakes or miss something
2. Cannot deploy same code in different environments.
  - Example: a "test" server and "production" server
3. Possibly insecure
  - exposes user/password, OAuth credentials, etc.

# Where to Put Configuration Data?

---

## 1. In a file.

Properties file (plain text) or similar  
XML or JSON file

## 2. In the environment.

Set env vars manually or using a Script  
Cloud services like Heroku have web form for this.

# What About Django?

**The Good:** All the configuration data is in one file

**The Bad:** Config is in code. You have to modify it for each different deployment. Can't check it in to Github.

```
import os, sys
SECRET_KEY = 'wjtc3c@k5m!3^0m3dq=e^jff_t%q*b1m'

DEBUG = True

ALLOWED_HOSTS = ['*']

INSTALLED_APPS = [
    'polls',
    'django.contrib.admin',
    'django.contrib.auth',
    ...
]
```

# Exercise

---

Look in your own `settings.py` file.

Find at least 4 settings that are either:

- 1) **confidential** - should not be visible to others
- 2) **may need to change** for different deployments, such as running on your own computer vs a server

# The 12-Factor App

---

Heroku recommended 12 characteristics of maintainable web applications.

**#3. Store configuration in the environment**

# 3. Store Config in the environment

---

"Config" is everything that is likely to vary between deployments (staging, production, local dev env.).

database handles: DATABASE\_URL = ...

credentials for other services your app uses

anything likely to change

OK to use a **configuration file** instead of environment...  
if there is a way to specify a different configuration file w/o changing the code.



# Django Example

Original settings.py:

```
SECRET_KEY = 'wjtc3c@k5m!3^0m3dq=e'
```

1. Store the secret key in the environment:

```
# Linux
export SECRET_KEY='wjtc3c@k5m!3^0m3dq=e'
# Windows:
set SECRET_KEY = 'wjtc3c@k5m!3^0m3dq=e'
```

2. Modified settings.py:

```
import os
SECRET_KEY = os.getenv('SECRET_KEY')
```

# Using python-decouple package

python-decouple has a 'config' function that is more flexible.  
In settings.py:

```
from decouple import config

SECRET_KEY = config('SECRET_KEY', default="secret")

DEBUG = config('DEBUG', cast=boolean)
```

`config()` will set 'SECRET\_KEY' and 'DEBUG' using either **environment variables** or values in a **file** named `.env`.

For example:

```
# .env file. values do not need quotes.
SECRET_KEY = wjtc3c@k5m!3^0m3dq=e
DEBUG = false
```

# Database Connection Info

Before externalization:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'polls',  
        'USER': 'padmin',  
        'PASSWORD': 'secret',  
        'HOST': 'localhost',  
        'PORT': 5432,  
    }  
}
```

# Externalize Database Connection

Using "config" for database parameters is clumsy:

```
DATABASES = {
    'default': {
        'ENGINE': config('DB_ENGINE'),
        'NAME': config('DB_NAME'),
        'USER': config('DB_USER'),
        'PASSWORD': config('DB_PASSWORD'),
        'HOST': config('DB_HOST'),
        'PORT': config('DB_PORT', cast=int),
    }
}
```

# File: .env

```
DB_ENGINE = django.db.backends.postgresql
DB_NAME = polls
DB_USER = padmin
DB_PASSWORD = secret
DB_HOST = 211.compute-1.amazonaws.com
DB_PORT = 5432
```

# `dj_database_url.parse()`

Use `dj_database_url` to create Django database parameters (dict) from a URL. Much simpler.

In `settings.py`:

```
DATABASES = {  
    'default': config('DATABASE_URL',  
                     cast=dj_database_url.parse)  
}
```

In the `.env` file or environment you write a single URL:

```
# .env file.  
DATABASE_URL=postgres://padmin:secret@localhost:5432/polls
```

# Properties File (Java example)

Plain text file containing lines of the form:

```
key = value
```

You do not need quotes around the value

Commonly used in many programming languages.

```
# Don't commit this file to Git!
jdbc.url =
    jdbc:mysql://cloud.google.com/xxxx
jdbc.user = pollsadmin
jdbc.password = secret
```

# Read a Properties File (Java)

Reading a properties file creates a *dictionary* or *map* of keys to values.

In Java:

```
InputStream in = new FileInputStream("myapp.conf");
Properties props = new Properties();
props.load(in);    // read properties from the file
// Get the database url using its key
// in the properties file
String url = props.getProperty("jdbc.url");
System.out.println("The database URL is " + url);
```

# Reference

---

*Externalize your Configuration*

`https://reflectoring.io/externalize-configuration/`

*The 12-Factor App* by Heroku, #3: Config

`https://12factor.net/`