IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

DJANGO ARCHITECTURE, CONFIGURATION FOR PRODUCTION SERVER

Nivid Dipakkumar Koradiya
PIET-IMCA Parul University
Vadodara, Gujarat

Abstract : This Paper is based on the methodologies used for deploying a Django app on production server. It covers all the aspects and methods prevailing in current market for developing a server/client ready application. Making a application production ready is important as the application in local machine and remote machine act very otherwise. So, we will be discussing various ways in which we should be structuring the Django application for better security and major.

***IndexTerms* - Django, Production Level integration, Deployment, MVT Structure, Models, API, Debug, Database, Cloud.**

I. INTRODUCTION

In the following sections, the paper will give you an overview of Django framework and its architecture, as well as a short history of its development. The paper will then examine the deployment process for Django apps, including the use of famous web services such as Apache and Nginx, as well as it will also delve into the different methods for scaling Django applications, including load balancing, caching, and database connectivity. This will include a discussion of the advantages and disadvantages of different deployment configurations and the trade-offs involved. In addition to deployment and configuration, the paper will also examine the crucial aspect of security in Django server environments. This will include a discussion of common security threats on applications and best practices for preventing and dodging these risks. In addition, the paper will also note a set of guidelines for future research in the area of Django production. In conclusion, this paper will serve as a valuable resource for organizations and developers looking to implement and sustain Django applications on servers. The paper's findings will provide practical guidance for students and developers looking to deploy and maintain high quality, scalable, and secure Django applications on servers.

II. HISTORY

Django an open to public web framework on Python that was first released in 2005. It was developed by a group of developers at the Lawrence Journal-World newspaper in Kansas, USA to ease out the process of building complex web applications. The framework went trending among developers for its ease of use and extremely powerful features, such as its emphasis on convention over configuration, keeping it easier and faster to build web applications. Over the time, Django has continued to evolve and has become widely adopted framework for building web applications. Today, Django is powering some of the largest and most complex websites in the world like Spotify, YouTube, NASA Etc.

III. URL ROUTING

URL routing is the procedure of mapping URLs to views. In other words, it allows you to connect a specific URL pattern with a view function that will handle the incoming request. This way, you can redirect different requests to different parts of web application. Django uses RE (regular expressions) to match URLs to view functions. We define your URL patterns in a file called urls.py, that is included in your Django app. The URL patterns are written using the path or re-path methods included in Django package itself, they take two arguments: the URL pattern and the view which should handle the incoming request.

IV. DEPLOYMENT SETTINGS

Django deployment settings are the configurations specific to a production environment of a Django web application. These settings may consist of database settings, security settings, mail settings, caching etc.

A. DEBUG settings:

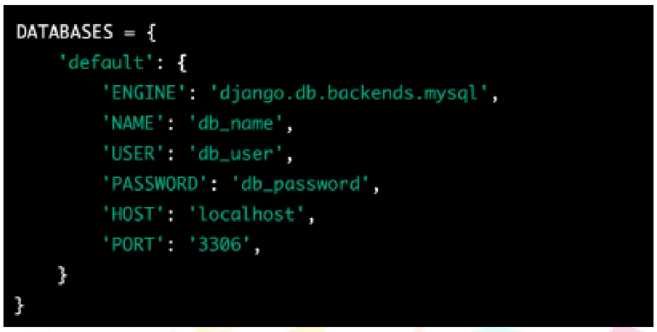
It Controls whether or not the developed application is running in debug mode. It should be set to False in production or deployment environments. If its running in the debug mode, this might create a security issue as it might reveals the code structure, patterns in which security is manage for an application.

B. ALLOWED_HOSTS settings:

This is a list of strings indicating the host/domain names that this Django site is required to serve. Also, it indicates the ip addresses that are allowed to make request and the server is liable to process those requests coming from the host mentioned in the list of allowed hosts.

C. DATABASES:

Databases is a dictionary consisting the settings for all databases being used with concerned application. It has all the settings including the host location of the database server, username and password credentials being used to access the database server, also the ports serving the server is mentioned here. A live example is given in next section:



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'db_name',
        'USER': 'db_user',
        'PASSWORD': 'db_password',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Figure -1 Database configuration in Django

V. STATIC FILES SETTINGS:

Static files in Django means files such as assets, JavaScript, and CSS files that are used to improve the visual experience and interactivity of app. One could use white noise library to serve the static files internally without needing an additional server. Two required config for static files are given below:

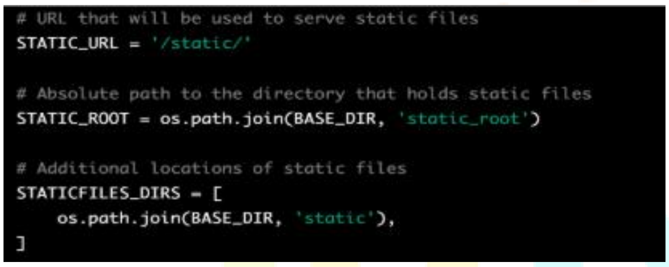
A. STATIC_URL:

It is a setting that indicates URL that will be allowed to serve the static files in a web application. So, by requesting the URL we could easily access the app's assets.

B. STATIC_ROOT:

This is the setting that specifies that from where the specific static files will be fetched when running on the deployment server. As well as it defined that when collect static command is executed then where do the files get dumped.[2].

For serving static files at a production level server, it's recommended to configure a additional web server namely Nginx or Apache for serving these static files, rather than serving them through the Django's dev. server.

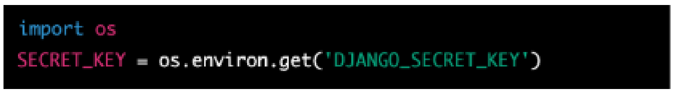


```
# URL that will be used to serve static files
STATIC_URL = '/static/'

# Absolute path to the directory that holds static files
STATIC_ROOT = os.path.join(BASE_DIR, 'static_root')

# Additional locations of static files
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]
```

Figure -2 Static files configuration

VI. SECRET KEY CONFIG AND STORAGE

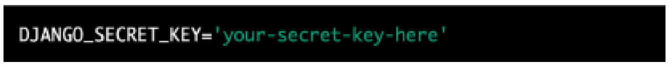
The secret key is used to apply the cryptographic signing, and always be kept secret. One should not hard-code key in your codebase, but should store it in an environment file. This Secret key is used to make passwords, exchange the encrypted info to and from the server's important middleware's. Example to import the key from environment and apply it to the application's setting:



```
import os
SECRET_KEY = os.environ.get('DJANGO_SECRET_KEY')
```

Figure -3 Accessing the env variables

To Load the secret key to the environment we would be using the .env file to store the important secure keys.



```
DJANGO_SECRET_KEY='your-secret-key-here'
```

Figure -4 Setting the Django secret key

Every time you change the .env file you should restart the server to load the modified settings from env file to environment variables.

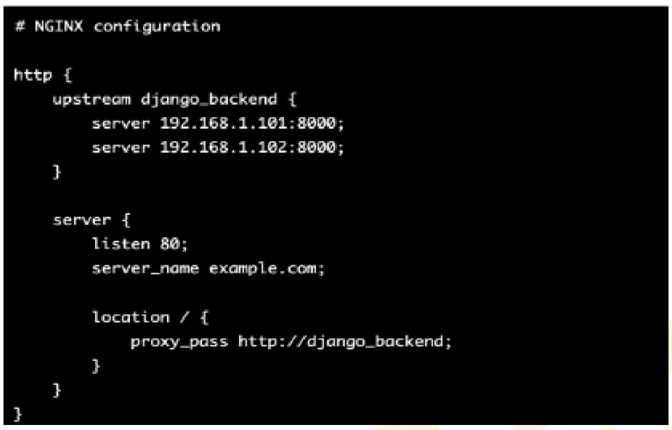
VII. LOAD BALANCING

Load balancing is a way used to distribute incoming requests across multiple web servers to ensure that no single production server is overloaded / bombarded with too much traffic. This can improve the overall performance and reliability of application by evenly distributing the incoming load across multiple available resources.

A.THE SOFTWARE LOAD BALANCER:

SB such as NGINX, can be used to allow incoming traffic to multiple Django servers. This is a preferable solution for lowertraffic environments or testing.

Here is how we would solve the load balancing for an application by running 2 different servers with IP allocated as shown in figure:



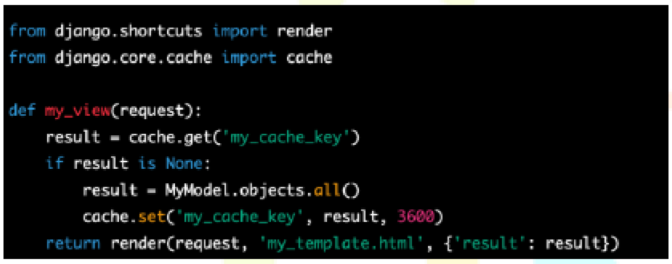
```
# NGINX configuration

http {
    upstream django_backend {
        server 192.168.1.101:8000;
        server 192.168.1.102:8000;
    }

    server {
        listen 80;
        server_name example.com;

        location / {
            proxy_pass http://django_backend;
        }
    }
}
```

Figure -5 Nginx Setting for load balancing

VIII. CACHING

Caching [3] is a technique employed to temporarily store frequently accessed assets in RAM or on storage-disk, so that following requests for the same data can be served instantly. This can greatly improve the efficiency of a server by lowering the time and resources needed to process each request.

To initiate caching in Django, you can use the cache framework present in django.core.cache package. It allows you to easily set up and use caching in your production application, and can carry multiple cache backends, including file-based caching, database caching, custom caching and memory-based caching. One simple example for caching a query set results in memory:



```
from django.shortcuts import render
from django.core.cache import cache

def my_view(request):
    result = cache.get('my_cache_key')
    if result is None:
        result = MyModel.objects.all()
        cache.set('my_cache_key', result, 3600)
    return render(request, 'my_template.html', {'result': result})
```

Figure -6 Caching in production server

In this above snippet, the view first confirms the cache for a value associated to the key 'my_cache_key'. If this value is not observed in the cache, the view fetches the result of the database query via the MyModel.objects.all() method, and pushes the result in the cache using the cache.set() method. The cached data is kept in storage for 3600 seconds that is 1 hour.

IX.SECURING THE SERVER

Securing a production server includes several steps to ensure the confidentiality, integrity, and availability of the API data and systems involved. Some best step for securing Django production servers are mentioned here:

A.Regular software updates:

Regularly updating the software and packages associated with the applications to ensure there are no risks that might arise from the older vulnerable codes.

B. Secure Authentication:

Taking adequate measures to protect the server's integrity including the 2FA – two factor authentication on sensitive parts of the application.

C. Using secure network protocols:

Adding network security is given a first place, we must protect the server network and the application using various firewall techniques, adding SSL in communication portion.

D. Frequent data backups:

Frequently taking the backup of data can be found beneficial in cases of server crash or integrity issue. The backup can be then restored with minimal amount of data loss.

X. PROTECTION AGAINST ATTACKS

A. SQL Injection:

This is a kind of attack, here an attacker injects malicious SQL code into app's SQL query, allowing them the access to manipulate important data kept in a database server.
To avoid SQL injection,[1] you must validate input and sanitize it prior using it in SQL based queries. We can also implement the ORM to write database queries, as it subconsciously escapes user input to prevent SQL injection attacks.

XI.REFERENCES

[1] L. Ma, D. Zhao, Y. Gao and C. Zhao, "Research on SQL Injection Attack and Prevention Technology Based on Web," 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA), Xi'an, China, 2019, pp. 176-179, doi: 10.1109/ICCNEA.2019.00042.

[2] Rubio, D. (2017). REST Services with Django. In: Beginning Django. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-2787-9_12

[3] S. Liawatimena et al., "Django Web Framework Software Metrics Measurement Using Radon and Pylint," 2018 Indonesian Association for Pattern Recognition International Conference (INAPR), Jakarta, Indonesia, 2018, pp. 218-222, doi: 10.1109/INAPR.2018.8627009.

[4] J. Chou, L. Chen, H. Ding, J. Tu and B. Xu, "A Method of [4] J. Chou, L. Chen, H. Ding, J. Tu and B. Xu, "A Method of Information System and Application Conference, Yangzhou, China, 2013, pp. 176-179, doi: 10.1109/WISA.2013.41.

[5] https://steelkiwi.com/blog/why-django-best-webframework-your-project/

[6] Prof. B Nithya Ramesh, Aashay R Amballi, and Vivekananda Mahanta,"Django the Python web framework" International Journal of Computer Science and Information Technology Research ISSN 2348- 120X

[7] https://docs.djangoproject.com/en/2.2/