1. Platform.sh

部署 Django 项目到 **Platform.sh** 需要按照以下步骤进行：

**1. 先确保本地环境正常**

在部署之前，确保你的 Django 项目在本地能够正常运行，并且已经完成了所有开发工作：

python manage.py runserver

如果本地一切正常，就可以进行部署。

**2. 创建 Platform.sh 账号并安装 CLI**

**（1）注册 Platform.sh 账号**

访问 [Platform.sh 官网](https://platform.sh/) 并注册一个账号。

**（2）安装 Platform.sh CLI**

在本地安装 Platform.sh CLI（命令行工具）：

curl -fsSL https://platform.sh/cli/installer | php

安装后，使用以下命令检查 CLI 是否安装成功：

platform --version

**（3）登录 CLI**

platform login

然后按照提示输入你的 Platform.sh 账号信息。

**3. 创建一个 Platform.sh 项目**

如果你还没有创建项目，可以在 Platform.sh 控制台上创建一个新的项目，或者使用 CLI 创建：

platform project:create

然后按照提示选择合适的方案。

**4. 在 Django 项目中配置 Platform.sh**

**（1）初始化 Platform.sh**

在 Django 项目根目录下运行：

platform init

这将会创建 .platform 目录，并包含默认的 services.yaml 和 routes.yaml 配置文件。

**5. 配置 .platform 目录**

Platform.sh 需要一些特殊的配置文件，你需要在项目根目录下创建 .platform 目录，并添加以下文件：

**（1）创建 .platform/services.yaml**

这个文件用于定义数据库、缓存等服务：

db:

type: postgresql:15

disk: 1024

如果你用的是 **MySQL**，可以改成：

db:

type: mysql:8.0

disk: 1024

**（2）创建 .platform/routes.yaml**

这个文件用于定义路由规则：

"https://{default}/":

type: upstream

upstream: "app:http"

**（3）创建 .platform.app.yaml**

这个文件是 Django 项目的主要配置：

name: app

type: python:3.11 # 选择合适的 Python 版本

dependencies:

python:

- gunicorn

- django

- psycopg2 # 如果你用 PostgreSQL

hooks:

build: |

pip install -r requirements.txt

deploy: |

python manage.py migrate

python manage.py collectstatic --noinput

mounts:

"/var": { "source": "data", "source\_path": "var" }

relationships:

database: "db:postgresql"

web:

commands:

start: "gunicorn myproject.wsgi --bind 0.0.0.0:8000"

**注意**：

* myproject.wsgi 需要替换成你的项目名。
* psycopg2 仅适用于 **PostgreSQL**，如果用 **MySQL**，换成 mysqlclient。

**6. 配置 Django**

**（1）修改 settings.py**

在 settings.py 里，添加数据库配置，让它从环境变量读取：

import os

import dj\_database\_url

DATABASES = {

'default': dj\_database\_url.config(default=os.getenv("DATABASE\_URL"))

}

ALLOWED\_HOSTS = ['\*'] # 或者指定 Platform.sh 提供的域名

**注意**：

* dj\_database\_url 需要安装：
* pip install dj-database-url
* ALLOWED\_HOSTS 建议写成 ['your-platformsh-domain.com']，你可以在 Platform.sh 控制台查看你的域名。

**7. 提交到 Git 并推送到 Platform.sh**

Platform.sh 使用 **Git 部署**，确保你的项目是 Git 仓库：

git init

git add .

git commit -m "Initial commit for Platform.sh deployment"

然后 **添加 Platform.sh 远程仓库**：

platform project:set-remote --project PROJECT\_ID

你可以在 Platform.sh 控制台找到你的 PROJECT\_ID。

最后，推送代码：

git push platform main

推送后，Platform.sh 会自动构建并部署你的 Django 项目。

**8. 访问网站**

部署完成后，你可以运行：

platform environment:routes

这会返回你的 **网站地址**，你可以在浏览器中打开这个地址，检查 Django 项目是否运行正常。

**9. 配置管理后台**

如果你要访问 Django 管理后台（/admin/），需要创建超级用户：

platform ssh

python manage.py createsuperuser

然后输入 **用户名、邮箱、密码**，创建管理员账号。

**10. 配置静态文件**

Platform.sh 不允许直接存储静态文件，你需要用 whitenoise 处理：

pip install whitenoise

然后在 settings.py 里添加：

MIDDLEWARE = [

"django.middleware.security.SecurityMiddleware",

"whitenoise.middleware.WhiteNoiseMiddleware", # 添加这一行

"django.contrib.sessions.middleware.SessionMiddleware",

# 其他中间件...

]

STATICFILES\_STORAGE = "whitenoise.storage.CompressedManifestStaticFilesStorage"

然后重新推送代码：

git add .

git commit -m "Add whitenoise for static files"

git push platform main

**总结**

1. **注册并创建 Platform.sh 项目**
2. **安装 Platform.sh CLI 并初始化项目**
3. **创建 .platform 目录，配置 services.yaml、routes.yaml 和 app.yaml**
4. **修改 settings.py 让 Django 适配 Platform.sh**
5. **使用 Git 推送代码到 Platform.sh**
6. **访问网站并测试**
7. **创建超级用户，配置管理后台**
8. **使用 whitenoise 处理静态文件**

这样你的 Django 项目就成功部署到 Platform.sh 了！🚀 如果有问题，欢迎继续问！

以上方法验证失败，国内需要开通信用卡，使用以下方法。

1. 把项目部署到linux服务器上，以腾讯云轻量应用服务器为例。

将 Django 项目部署到 Debian 服务器通常需要以下步骤，涵盖环境配置、Web 服务器部署和安全优化。以下是详细操作指南：

---

### \*\*一、准备工作\*\*

#### 1. \*\*连接到服务器\*\*

```bash

ssh root@服务器IP # 使用 SSH 登录服务器（建议创建非 root 用户操作）

```

把项目放到github上做中转，采用如下方法。

先到github上新建一个仓库，然后运行如下命令：

echo "# pizzeria\_project" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/yangguohuan/pizzeria\_project.git

git push -u origin main

#### 2. \*\*更新系统和安装基础工具\*\*

```bash

sudo apt update && sudo apt upgrade -y

sudo apt install -y curl wget git vim ufw

```

---

### \*\*二、安装 Python 环境\*\*

#### 1. \*\*安装 Python 和 pip\*\*

```bash

sudo apt install -y python3 python3-pip python3-venv

```

#### 2. \*\*创建 Python 虚拟环境\*\*

```bash

cd /opt

sudo mkdir myproject && cd myproject

python3 -m venv venv # 创建虚拟环境

source venv/bin/activate # 激活虚拟环境

```

---

### \*\*三、部署 Django 项目\*\*

#### 1. \*\*上传项目代码\*\*

- 通过 Git 克隆代码（或手动上传）：

```bash

git clone https://github.com/yourusername/yourproject.git

cd yourproject

```

#### 2. \*\*安装项目依赖\*\*

```bash

pip install -r requirements.txt

```

#### 3. \*\*配置数据库\*\*

- 安装 PostgreSQL（推荐）：

```bash

sudo apt install -y postgresql postgresql-contrib

sudo -u postgres psql

```

- 在 PostgreSQL 中创建数据库和用户：

```sql

CREATE DATABASE mydb;

CREATE USER myuser WITH PASSWORD 'mypassword';

GRANT ALL PRIVILEGES ON DATABASE mydb TO myuser;

\q

```

#### 4. \*\*修改 Django 配置\*\*

- 编辑 `settings.py`：

```python

DEBUG = False

ALLOWED\_HOSTS = ['yourdomain.com', '服务器IP'] # 替换为实际域名或IP

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'mydb',

'USER': 'myuser',

'PASSWORD': 'mypassword',

'HOST': 'localhost',

'PORT': '5432',

}

}

STATIC\_URL = '/static/'

STATIC\_ROOT = '/opt/myproject/static/' # 静态文件收集路径

```

#### 5. \*\*迁移数据库和收集静态文件\*\*

```bash

python manage.py migrate

python manage.py collectstatic

```

---

### \*\*四、配置 Gunicorn 作为应用服务器\*\*

#### 1. \*\*安装 Gunicorn\*\*

```bash

pip install gunicorn

```

#### 2. \*\*创建 Gunicorn 服务文件\*\*

- 新建文件 `/etc/systemd/system/gunicorn.service`：

```ini

[Unit]

Description=gunicorn daemon

After=network.target

[Service]

User=root

Group=www-data

WorkingDirectory=/opt/myproject/yourproject

ExecStart=/opt/myproject/venv/bin/gunicorn \

--access-logfile - \

--workers 3 \

--bind unix:/opt/myproject/gunicorn.sock \

yourproject.wsgi:application

[Install]

WantedBy=multi-user.target

```

#### 3. \*\*启动 Gunicorn\*\*

```bash

sudo systemctl daemon-reload

sudo systemctl start gunicorn

sudo systemctl enable gunicorn

```

---

### \*\*五、配置 Nginx 作为反向代理\*\*

#### 1. \*\*安装 Nginx\*\*

```bash

sudo apt install -y nginx

```

#### 2. \*\*创建 Nginx 配置文件\*\*

- 新建文件 `/etc/nginx/sites-available/myproject`：

```nginx

server {

listen 80;

server\_name yourdomain.com 服务器IP;

location /static/ {

alias /opt/myproject/static/;

}

location / {

proxy\_set\_header Host $http\_host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_pass http://unix:/opt/myproject/gunicorn.sock;

}

}

```

#### 3. \*\*启用配置并重启 Nginx\*\*

```bash

sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled/

sudo nginx -t # 测试配置语法

sudo systemctl restart nginx

```

---

### \*\*六、安全优化\*\*

#### 1. \*\*配置防火墙\*\*

```bash

sudo ufw allow 80/tcp # HTTP

sudo ufw allow 22/tcp # SSH

sudo ufw enable

```

#### 2. \*\*配置 HTTPS（可选）\*\*

- 使用 Certbot 获取免费 SSL 证书：

```bash

sudo apt install -y certbot python3-certbot-nginx

sudo certbot --nginx -d yourdomain.com

```

#### 3. \*\*关闭 DEBUG 模式\*\*

- 确保 `settings.py` 中 `DEBUG = False`，并配置日志记录。

---

### \*\*七、验证部署\*\*

1. 访问 `http://服务器IP` 或 `http://yourdomain.com`，检查 Django 项目是否正常运行。

2. 查看日志排查问题：

```bash

sudo journalctl -u gunicorn # Gunicorn 日志

sudo tail -f /var/log/nginx/error.log # Nginx 错误日志

```

---

### \*\*八、常见问题解决\*\*

- \*\*静态文件 404 错误\*\*：确保 `STATIC\_ROOT` 路径正确且 Nginx 配置的 `alias` 指向该路径。

- \*\*数据库连接失败\*\*：检查 PostgreSQL 用户权限和 `settings.py` 中的数据库配置。

- \*\*Gunicorn 服务未启动\*\*：运行 `sudo systemctl status gunicorn` 查看错误信息。

---

通过以上步骤，你的 Django 项目已成功部署到 Debian 服务器。后续可进一步优化：

- 使用 Docker 容器化部署。

- 配置 CI/CD 自动化流程（如 GitHub Actions）。

- 监控服务器性能（如 Prometheus + Grafana）。