کاملاً متوجه هستم! دو نکته مهم را باید در نظر بگیریم: **استفاده از MySQL به جای PostgreSQL** و **مشکلات دسترسی به مخازن لینوکس به دلیل تحریم‌ها**.

با توجه به این موارد، راهنمای قبلی را با تغییرات لازم و راهکارهایی برای دور زدن محدودیت‌های تحریم بازنویسی می‌کنم.

**سناریو: پیاده‌سازی هاست جنگو روی VMware vSphere با MySQL (پشتیبانی از تحریم)**

همچنان فرض می‌کنیم شما در VMware vSphere یک **ماشین مجازی (VM)** جدید با سیستم عامل **Ubuntu Server LTS** (مثلاً Ubuntu 22.04 LTS) ایجاد کرده‌اید و به آن دسترسی SSH دارید.

**مراحل کلی با توجه به شرایط جدید:**

1. **آماده‌سازی ماشین مجازی و حل مشکل دسترسی به مخازن:** تنظیم منابع جایگزین (Mirror) برای apt یا استفاده از روش‌های دیگر.
2. **نصب و پیکربندی MySQL:** راه‌اندازی دیتابیس برای جنگو.
3. **پیکربندی کاربر سیستم برای جنگو:** ایجاد یک کاربر غیر root برای اجرای امن برنامه.
4. **کپی کردن و آماده‌سازی پروژه جنگو:** انتقال کد، نصب وابستگی‌ها و تنظیمات پروداکشن.
5. **پیکربندی Gunicorn:** سرور WSGI که اپلیکیشن جنگو را اجرا می‌کند.
6. **پیکربندی Nginx:** وب سرور که درخواست‌ها را مدیریت کرده و به Gunicorn ارجاع می‌دهد.
7. **پیکربندی فایروال:** باز کردن پورت‌های لازم.
8. **آزمایش نهایی:** اطمینان از عملکرد صحیح برنامه.

**بخش اول: آماده‌سازی ماشین مجازی و حل مشکل دسترسی به مخازن**

برای حل مشکل دسترسی به مخازن (Repositories) اوبونتو به دلیل تحریم‌ها، می‌توانید از **Mirrorهای داخلی ایران** یا **Mirrorهای عمومی که فیلتر نیستند** استفاده کنید.

1. اتصال به VM از طریق SSH:

با استفاده از SSH به VM خود متصل شوید:

Bash

ssh username@your\_vm\_ip\_address

1. تغییر Mirrorهای APT:

این مهمترین گام برای حل مشکل تحریم است. فایل sources.list را ویرایش می‌کنیم:

Bash

sudo nano /etc/apt/sources.list

تمام خطوطی که با deb و deb-src شروع می‌شوند را با گذاشتن # در ابتدای آن‌ها کامنت (Comment) کنید.

سپس، خطوط مربوط به یک Mirror داخلی یا عمومی که قابل دسترسی است را به انتهای فایل اضافه کنید. برای مثال، می‌توانید از Mirror دانشگاه فردوسی مشهد استفاده کنید (یا جستجو کنید "ubuntu apt mirror iran" برای یافتن موارد دیگر):

# Original Ubuntu repositories (commented out due to sanctions)

# deb http://archive.ubuntu.com/ubuntu/ jammy main restricted universe multiverse

# deb http://archive.ubuntu.com/ubuntu/ jammy-updates main restricted universe multiverse

# deb http://archive.ubuntu.com/ubuntu/ jammy-backports main restricted universe multiverse

# deb http://security.ubuntu.com/ubuntu/ jammy-security main restricted universe multiverse

# Recommended Mirror for Iran (e.g., Ferdowsi University of Mashhad)

deb http://mirror.fardis.io/ubuntu/ jammy main restricted universe multiverse

deb http://mirror.fardis.io/ubuntu/ jammy-updates main restricted universe multiverse

deb http://mirror.fardis.io/ubuntu/ jammy-backports main restricted universe multiverse

deb http://mirror.fardis.io/ubuntu/ jammy-security main restricted universe multiverse

# Or any other accessible mirror. You might need to test a few.

**نکته:** jammy نام کد (Codename) برای Ubuntu 22.04 LTS است. اگر از نسخه دیگری از اوبونتو استفاده می‌کنید، آن را با Codename نسخه خود جایگزین کنید.

با Ctrl+X، سپس Y و Enter فایل را ذخیره و خارج شوید.

1. به‌روزرسانی سیستم عامل با Mirror جدید:

حالا با Mirror جدید، لیست بسته‌ها را به‌روز کرده و بسته‌های موجود را ارتقا دهید:

Bash

sudo apt update

sudo apt upgrade -y

این دستور باید بدون مشکل اجرا شود. اگر خطا گرفتید، احتمالاً Mirror انتخابی شما نیز مسدود است و باید یک Mirror دیگر را امتحان کنید.

1. نصب پکیج‌های ضروری:

برای راه‌اندازی جنگو و ابزارهای مرتبط، به بسته‌های زیر نیاز داریم:

Bash

sudo apt install python3-pip python3-dev libmysqlclient-dev nginx curl git -y

* + libmysqlclient-dev: این پکیج شامل کتابخانه‌های توسعه MySQL است که برای اتصال پایتون به MySQL (از طریق mysqlclient یا PyMySQL که در جنگو استفاده می‌شود) لازم است.

**بخش دوم: نصب و پیکربندی MySQL Server**

در این سناریو، دیتابیس را روی همین VM نصب می‌کنیم.

1. **نصب MySQL Server:**

Bash

sudo apt install mysql-server -y

در حین نصب، ممکن است از شما خواسته شود یک رمز عبور برای کاربر root مای‌اس‌کیوال تعیین کنید. این رمز عبور را به خاطر بسپارید.

1. ایمن‌سازی نصب MySQL:

بعد از نصب، ابزار ایمن‌سازی MySQL را اجرا کنید تا نصب خود را ایمن کنید:

Bash

sudo mysql\_secure\_installation

* + **Validate Password Component:** این را Y یا n انتخاب کنید. اگر Y انتخاب کردید، قدرت رمز عبور جدید شما سنجیده می‌شود.
  + **Change the password for root?**: اگر قبلاً رمز عبور تعیین نکردید یا می‌خواهید تغییر دهید، Y را بزنید.
  + **Remove anonymous users?**: Y را بزنید.
  + **Disallow root login remotely?**: Y را بزنید. این از دسترسی root مای‌اس‌کیوال از راه دور جلوگیری می‌کند که امن‌تر است.
  + **Remove test database and access to it?**: Y را بزنید.
  + **Reload privilege tables now?**: Y را بزید.

1. ایجاد دیتابیس و کاربر جدید برای جنگو در MySQL:

وارد محیط خط فرمان مای‌اس‌کیوال شوید (با کاربر root مای‌اس‌کیوال و رمز عبوری که تعیین کردید):

Bash

sudo mysql -u root -p

رمز عبور root مای‌اس‌کیوال را وارد کنید. حالا در خط فرمان mysql> دستورات زیر را وارد کنید:

SQL

CREATE DATABASE myprojectdb CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

CREATE USER 'myprojectuser'@'localhost' IDENTIFIED BY 'your\_secure\_password'; -- رمز عبور قوی انتخاب کنید

GRANT ALL PRIVILEGES ON myprojectdb.\* TO 'myprojectuser'@'localhost';

FLUSH PRIVILEGES;

EXIT;

* + myprojectdb: نام دیتابیس شما.
  + myprojectuser: نام کاربری که جنگو با آن به دیتابیس متصل می‌شود.
  + your\_secure\_password: **رمز عبور قوی** برای کاربر دیتابیس جنگو.
  + CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci: برای پشتیبانی کامل از کاراکترهای یونیکد (از جمله فارسی و ایموجی‌ها) بسیار توصیه می‌شود.

**بخش سوم: پیکربندی کاربر سیستم برای جنگو**

این مرحله مانند قبل است و برای افزایش امنیت، برنامه جنگو را با یک کاربر غیر root اجرا می‌کنیم.

1. **ایجاد کاربر جدید:**

Bash

sudo adduser --system --no-create-home django\_user

1. **اضافه کردن کاربر به گروه** www-data**:**

Bash

sudo usermod -a -G www-data django\_user

**بخش چهارم: کپی کردن و آماده‌سازی پروژه جنگو**

در این مرحله، پروژه جنگو خود را به سرور منتقل می‌کنیم و آن را برای محیط پروداکشن آماده می‌سازیم.

1. مکان پروژه و تنظیم مجوزها:

یک دایرکتوری مناسب برای پروژه خود ایجاد کرده و مالکیت آن را به کاربر django\_user دهید:

Bash

sudo mkdir -p /opt/myproject

sudo chown django\_user:www-data /opt/myproject

1. انتقال کد پروژه:

از طریق git clone یا scp کد پروژه را به /opt/myproject منتقل کنید. (مثلاً در دایرکتوری /opt/myproject/myproject\_code).

Bash

sudo -u django\_user git clone https://your\_repository\_url /opt/myproject/myproject\_code

**نکته:** اگر git شما به مشکل پروکسی یا تحریم برخورد، می‌توانید فایل‌های پروژه را به صورت فشرده (مثلاً .zip یا .tar.gz) به سرور منتقل کرده و سپس از حالت فشرده خارج کنید.

1. ایجاد محیط مجازی پایتون:

وارد دایرکتوری کد پروژه شوید و محیط مجازی را ایجاد کنید:

Bash

cd /opt/myproject/myproject\_code

python3 -m venv venv

source venv/bin/activate

1. نصب وابستگی‌های پایتون:

در این مرحله، باید مطمئن شوید که در فایل requirements.txt شما، به جای psycopg2 یا psycopg2-binary (که برای PostgreSQL است)، از کتابخانه‌های اتصال به MySQL استفاده شده است.

توصیه می‌شود از mysqlclient یا PyMySQL استفاده کنید. اگر از mysqlclient استفاده می‌کنید، نیاز به libmysqlclient-dev که قبلاً نصب کردیم، خواهید داشت.

* + **اگر** mysqlclient **در** requirements.txt **شماست:**

Bash

pip install -r requirements.txt

pip install gunicorn

* + **اگر از** PyMySQL **استفاده می‌کنید (نیاز به** libmysqlclient-dev **ندارد):**

Bash

# اطمینان حاصل کنید PyMySQL در requirements.txt هست

pip install -r requirements.txt

pip install gunicorn

1. تنظیمات پروژه جنگو (settings.py):

مهمترین تغییرات در settings.py برای محیط پروداکشن:

* + DEBUG = False: این حیاتی است.
  + ALLOWED\_HOSTS: لیست آدرس‌های IP یا دامنه‌هایی که Nginx از طریق آن‌ها به برنامه شما دسترسی پیدا می‌کند.

Python

ALLOWED\_HOSTS = ['your\_server\_ip\_address', 'your\_domain.com', 'localhost']

* + DATABASES: تنظیمات دیتابیس MySQL که قبلاً ایجاد کردید:

Python

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql', # تغییر به mysql

'NAME': 'myprojectdb',

'USER': 'myprojectuser',

'PASSWORD': 'your\_secure\_password', # همان رمز عبوری که در بخش 2 تعیین کردید

'HOST': 'localhost',

'PORT': '3306',

}

}

**نکته:** اگر از PyMySQL استفاده می‌کنید، باید آن را به عنوان Engine به جنگو معرفی کنید:

Python

DATABASES = {

'default': {

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

'NAME': 'myprojectdb',

'USER': 'myprojectuser',

'PASSWORD': 'your\_secure\_password',

'HOST': 'localhost',

'PORT': '3306',

'OPTIONS': {

'read\_default\_file': '/etc/mysql/my.cnf', # optional, if you use a cnf file

'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'", # important for some setups

},

}

}

و در بالای settings.py اضافه کنید:

Python

import pymysql

pymysql.install\_as\_MySQLdb()

* + STATIC\_ROOT **و** MEDIA\_ROOT: (مانند قبل)

Python

import os

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'mediafiles')

MEDIA\_URL = '/media/'

* + SECRET\_KEY: (مانند قبل) از متغیر محیطی استفاده کنید.

1. اجرای collectstatic و migrate:

این دستورات را در حالی که در محیط مجازی پایتون هستید، اجرا کنید:

Bash

python manage.py collectstatic --noinput

python manage.py migrate

**بخش پنجم: پیکربندی Gunicorn**

این مرحله مانند قبل است، فقط مطمئن شوید WorkingDirectory و ExecStart به مسیر صحیح پروژه شما اشاره می‌کنند.

1. **ایجاد فایل Systemd برای Gunicorn:**

Bash

sudo nano /etc/systemd/system/gunicorn.service

محتوای زیر را در آن قرار دهید:

Ini, TOML

[Unit]

Description=Gunicorn instance to serve my Django project

After=network.target

[Service]

User=django\_user

Group=www-data

WorkingDirectory=/opt/myproject/myproject\_code

Environment="PATH=/opt/myproject/myproject\_code/venv/bin"

ExecStart=/opt/myproject/myproject\_code/venv/bin/gunicorn --workers 3 --bind unix:/run/gunicorn.sock your\_project\_name.wsgi:application

ExecReload=/bin/kill -s HUP $MAINPID

KillMode=mixed

TimeoutStopSec=5

PrivateTmp=true

[Install]

WantedBy=multi-user.target

* + your\_project\_name.wsgi:application: نام پکیج جنگو شما.

1. **فعال و راه‌اندازی Gunicorn:**

Bash

sudo systemctl daemon-reload

sudo systemctl start gunicorn

sudo systemctl enable gunicorn

sudo systemctl status gunicorn

**بخش ششم: پیکربندی Nginx**

این مرحله نیز مانند قبل است و نیازی به تغییر خاصی ندارد.

1. **ایجاد فایل پیکربندی Nginx برای پروژه شما:**

Bash

sudo nano /etc/nginx/sites-available/myproject

محتوای زیر را در آن قرار دهید:

Nginx

server {

listen 80;

server\_name your\_vm\_ip\_address your\_domain.com;

location /static/ {

alias /opt/myproject/myproject\_code/staticfiles/;

}

location /media/ {

alias /opt/myproject/myproject\_code/mediafiles/;

}

location / {

include proxy\_params;

proxy\_pass http://unix:/run/gunicorn.sock;

}

}

1. **فعال کردن پیکربندی Nginx:**

Bash

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

sudo rm /etc/nginx/sites-enabled/default

1. **بررسی Syntax و راه‌اندازی مجدد Nginx:**

Bash

sudo nginx -t

sudo systemctl restart nginx

**بخش هفتم: پیکربندی فایروال (UFW)**

این مرحله نیز مانند قبل است.

1. **اجازه دادن به اتصالات SSH، HTTP و HTTPS:**

Bash

sudo ufw allow 'OpenSSH'

sudo ufw allow 'Nginx HTTP'

sudo ufw allow 'Nginx HTTPS' # اگر از HTTPS استفاده می‌کنید

1. **فعال کردن فایروال:**

Bash

sudo ufw enable

**بخش هشتم: آزمایش نهایی و گام‌های تکمیلی**

1. آزمایش برنامه:

در مرورگر خود، آدرس http://your\_vm\_ip\_address را وارد کنید تا سایت جنگو خود را مشاهده کنید.

1. بررسی لاگ‌ها:

برای اشکال‌زدایی، لاگ‌های Gunicorn و Nginx را بررسی کنید:

Bash

sudo journalctl -u gunicorn --since "1 hour ago"

sudo tail -f /var/log/nginx/error.log

sudo tail -f /var/log/nginx/access.log

1. امن‌سازی با HTTPS:

به شدت توصیه می‌شود وب‌سایت خود را با HTTPS امن کنید. اگر Mirrorهای snap در دسترس باشند، می‌توانید از Certbot برای Let's Encrypt استفاده کنید:

Bash

sudo snap install core

sudo snap refresh core

sudo snap install --classic certbot

sudo ln -s /snap/bin/certbot /usr/bin/certbot

sudo certbot --nginx

اگر snap به دلیل تحریم‌ها مشکل داشت، راه‌حل‌های جایگزین برای SSL (مثل خرید گواهینامه از یک شرکت ایرانی یا استفاده از acme.sh که کمتر به ابزارهای سیستمی وابسته است) را در نظر بگیرید.

با اعمال این تغییرات، شما می‌توانید برنامه جنگو خود را با MySQL در محیط پروداکشن روی یک VM اوبونتو، حتی با وجود محدودیت‌های تحریم، راه‌اندازی کنید.

آیا در طول این مراحل با مشکل خاصی روبرو شدید یا نیاز به توضیح بیشتری در مورد هر بخش دارید؟

بسیار عالی! انتقال پروژه Django از محیط توسعه (Development) به محیط استقرار (Deployment/Production) یک سناریوی استاندارد در توسعه نرم‌افزار است و گام‌های مشخصی دارد. این فرآیند معمولاً شامل آماده‌سازی کد، تنظیمات سرور، و راه‌اندازی دائمی برنامه است.

سناریوی کامل برای استقرار یک پروژه Django به شرح زیر است:

**سناریوی استقرار پروژه Django (از PyCharm به Production)**

فرض می‌کنیم شما کد را در PyCharm توسعه داده‌اید و حالا می‌خواهید آن را روی یک سرور لینوکس (مثلاً Ubuntu) یا حتی یک سرور ویندوز (که البته لینوکس رایج‌تر است) مستقر کنید.

**گام 1: آماده‌سازی پروژه Django برای Production (در محیط توسعه)**

1. **بررسی** DEBUG = False **در** settings.py**:**
   * در فایل settings.py پروژه خود، مطمئن شوید که DEBUG به False تغییر کرده باشد:

Python

# settings.py

DEBUG = False

* + اگر DEBUG = True باشد، اطلاعات حساس پروژه (مثل خطاهای Traceback) به کاربران نمایش داده می‌شود که خطر امنیتی دارد.

1. **تنظیم** ALLOWED\_HOSTS**:**
   * در settings.py، لیست آدرس‌های IP یا نام‌های دامنه‌ای را که سرور شما به آن‌ها پاسخ می‌دهد، وارد کنید:

Python

# settings.py

ALLOWED\_HOSTS = ['your\_domain.com', 'www.your\_domain.com', 'your\_server\_ip\_address']

# برای تست اولیه می توانید ['\*'] قرار دهید، اما برای production امن نیست.

1. **تنظیمات دیتابیس:**
   * اگر در توسعه از SQLite استفاده می‌کردید، برای Production معمولاً به دیتابیس‌های قوی‌تری مانند PostgreSQL یا MySQL نیاز دارید. تنظیمات مربوط به دیتابیس جدید را در settings.py به‌روز کنید.
   * مثال برای PostgreSQL:

Python

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql\_psycopg2',

'NAME': 'your\_database\_name',

'USER': 'your\_database\_user',

'PASSWORD': 'your\_database\_password',

'HOST': 'localhost', # یا آدرس IP سرور دیتابیس

'PORT': '',

}

}

* + اطمینان حاصل کنید که پکیج مربوط به دیتابیس (مثلاً psycopg2-binary برای PostgreSQL) در requirements.txt شما وجود دارد.

1. **تنظیمات فایل‌های استاتیک و مدیا:**
   * STATIC\_ROOT را در settings.py تنظیم کنید. این مسیری است که فایل‌های استاتیک جمع‌آوری‌شده در آن قرار می‌گیرند.

Python

# settings.py

import os

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

STATIC\_URL = '/static/'

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles') # پوشه‌ای که فایل‌های استاتیک جمع‌آوری می‌شوند

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'mediafiles') # پوشه‌ای برای آپلودهای کاربر

* + **جمع‌آوری فایل‌های استاتیک:** در ترمینال PyCharm (در محیط مجازی فعال) این دستور را اجرا کنید:

Bash

python manage.py collectstatic

این کار تمام فایل‌های استاتیک شما را به پوشه STATIC\_ROOT منتقل می‌کند.

1. **ساخت** requirements.txt **نهایی:**
   * اطمینان حاصل کنید که تمام پکیج‌های مورد نیاز در requirements.txt ذخیره شده‌اند:

Bash

pip freeze > requirements.txt

1. **مدیریت متغیرهای محیطی حساس:**
   * اطلاعات حساسی مانند SECRET\_KEY، رمز عبور دیتابیس و API Keyها را مستقیماً در settings.py قرار ندهید. به جای آن، از متغیرهای محیطی (Environment Variables) یا فایل‌های .env استفاده کنید.
   * می‌توانید از پکیج python-dotenv برای این کار استفاده کنید:

Python

# settings.py

import os

from dotenv import load\_dotenv

load\_dotenv() # این خط را در بالای settings.py اضافه کنید

SECRET\_KEY = os.getenv('DJANGO\_SECRET\_KEY') # در فایل .env خود، DJANGO\_SECRET\_KEY='your\_secret\_key' قرار دهید

# ... و برای دیتابیس نیز

* + **حتماً فایل** .env **را به** .gitignore **اضافه کنید تا در سیستم کنترل نسخه (اگر استفاده می‌کنید) قرار نگیرد.**

**گام 2: انتقال پروژه به سرور**

1. **فشرده‌سازی پروژه:**
   * پوشه اصلی پروژه Django خود (که حالا شامل requirements.txt و پوشه staticfiles است) را به صورت .zip یا .tar.gz فشرده کنید. **پوشه** venv **را حذف کنید** تا حجم فایل کم شود.
2. **انتقال فایل فشرده به سرور:**
   * با استفاده از scp (برای لینوکس/مک) یا WinSCP/FileZilla (برای ویندوز) فایل فشرده را به سرور منتقل کنید.

Bash

# مثال با scp (در ترمینال سیستم لوکال شما)

scp myproject.zip user@your\_server\_ip:/path/to/destination/

**گام 3: تنظیم و راه‌اندازی در سرور**

1. **اتصال به سرور:**
   * با SSH به سرور خود متصل شوید.

Bash

ssh user@your\_server\_ip

1. **نصب پیش‌نیازها:**
   * اگر پایتون و pip نصب نیستند، آن‌ها را نصب کنید.
   * پکیج‌های مورد نیاز برای دیتابیس (مثلاً libpq-dev برای PostgreSQL) و Pillow (اگر از تصاویر استفاده می‌کنید) را نصب کنید.

Bash

# برای Ubuntu/Debian

sudo apt update

sudo apt install python3-pip python3-venv postgresql-client libpq-dev python3-dev build-essential

# اگر از MySQL استفاده می‌کنید:

# sudo apt install default-libmysqlclient-dev

1. **باز کردن فایل فشرده و ایجاد محیط مجازی:**
   * فایل فشرده را در مسیر مورد نظر (مثلاً /var/www/yourproject) باز کنید.

Bash

sudo mkdir -p /var/www/yourproject

sudo unzip myproject.zip -d /var/www/yourproject/ # یا tar -xzf

cd /var/www/yourproject/myproject\_folder/ # وارد پوشه اصلی پروژه شوید

* + یک محیط مجازی جدید بسازید و آن را فعال کنید:

Bash

python3 -m venv venv

source venv/bin/activate

1. **نصب وابستگی‌ها:**
   * کتابخانه‌های مورد نیاز را از requirements.txt نصب کنید:

Bash

pip install -r requirements.txt

1. **اعمال مهاجرت‌های دیتابیس (Migrations):**
   * اگر دیتابیس را تغییر داده‌اید یا دیتابیس جدیدی دارید، مهاجرت‌ها را اعمال کنید:

Bash

python manage.py migrate

* + اگر نیاز به ساخت Superuser دارید:

Bash

python manage.py createsuperuser

1. **تست اولیه پروژه (اختیاری):**
   * برای اطمینان از اینکه همه چیز درست است، می‌توانید سرور توسعه Django را برای مدت کوتاهی روی سرور اجرا کنید (فقط برای تست، نه Production):

Bash

python manage.py runserver 0.0.0.0:8000

* + با باز کردن آدرس IP سرور و پورت 8000 در مرورگر، می‌توانید پروژه را مشاهده کنید. (حتماً فایروال سرور را بررسی کنید).

**گام 4: راه‌اندازی Web Server و WSGI Server**

Django به تنهایی نمی‌تواند ترافیک HTTP را هندل کند. شما به یک Web Server (مثل Nginx یا Apache) و یک WSGI Server (مثل Gunicorn یا uWSGI) نیاز دارید.

1. **نصب WSGI Server (مثلاً Gunicorn):**

Bash

pip install gunicorn

1. **تست Gunicorn:**
   * وارد پوشه اصلی پروژه خود شوید (که manage.py در آن قرار دارد) و دستور زیر را اجرا کنید:

Bash

gunicorn your\_project\_name.wsgi:application --bind 0.0.0.0:8000

(به جای your\_project\_name، نام پوشه اصلی پروژه که settings.py در آن قرار دارد را بگذارید.)

* + با مرورگر به آدرس http://your\_server\_ip:8000 بروید تا مطمئن شوید Gunicorn کار می‌کند.

1. **نصب و تنظیم Web Server (مثلاً Nginx):**
   * Nginx درخواست‌های HTTP را دریافت کرده و آن‌ها را به Gunicorn (که برنامه Django شما را اجرا می‌کند) منتقل می‌کند. همچنین Nginx فایل‌های استاتیک و مدیا را به صورت مستقیم سرو می‌کند.

Bash

sudo apt install nginx

* + یک فایل تنظیمات Nginx برای پروژه خود در etc/nginx/sites-available/your\_project بسازید (یا فایل default را ویرایش کنید):

Nginx

server {

listen 80;

server\_name your\_domain.com www.your\_domain.com your\_server\_ip;

location = /favicon.ico { access\_log off; log\_not\_found off; }

location /static/ {

root /var/www/yourproject/myproject\_folder/; # مسیر STATIC\_ROOT شما

}

location /media/ {

root /var/www/yourproject/myproject\_folder/; # مسیر MEDIA\_ROOT شما

}

location / {

include proxy\_params;

proxy\_pass http://unix:/run/gunicorn.sock; # این سوکت را در مرحله بعدی می سازیم

}

}

* + یک لینک سیمبلیک به sites-enabled ایجاد کنید و Nginx را ری‌استارت کنید:

Bash

sudo ln -s /etc/nginx/sites-available/your\_project /etc/nginx/sites-enabled/

sudo nginx -t # تست تنظیمات

sudo systemctl restart nginx

1. **تنظیم Gunicorn با Systemd (برای اجرا در پس‌زمینه):**
   * برای اینکه Gunicorn به طور خودکار با راه‌اندازی سرور اجرا شود و در پس‌زمینه بماند، یک فایل سرویس Systemd بسازید:

Bash

sudo nano /etc/systemd/system/gunicorn.service

* + محتوای زیر را در آن قرار دهید (با توجه به مسیرهای پروژه و نام کاربری خود):

Ini, TOML

[Unit]

Description=Gunicorn daemon for your Django project

After=network.target

[Service]

User=your\_username # نام کاربری که با آن به سرور لاگین می کنید یا یک کاربر سیستم

Group=www-data # گروهی که Nginx استفاده می کند

WorkingDirectory=/var/www/yourproject/myproject\_folder/ # مسیر ریشه پروژه

ExecStart=/var/www/yourproject/myproject\_folder/venv/bin/gunicorn \

--access-logfile - \

--workers 3 \

--bind unix:/run/gunicorn.sock \

your\_project\_name.wsgi:application

[Install]

WantedBy=multi-user.target

* + سرویس Systemd را فعال و شروع کنید:

Bash

sudo systemctl start gunicorn

sudo systemctl enable gunicorn

sudo systemctl status gunicorn # برای بررسی وضعیت

**گام 5: امنیت (مهم!)**

1. **فایروال:**
   * پورت‌های ضروری (80 برای HTTP و 443 برای HTTPS) را باز کنید و بقیه را ببندید.
   * مثال با UFW در اوبونتو:

Bash

sudo ufw allow 'Nginx Full'

sudo ufw enable

1. **HTTPS با Let's Encrypt:**
   * برای رمزنگاری ارتباط و امنیت، حتماً HTTPS را فعال کنید. Certbot یک ابزار عالی برای دریافت گواهی رایگان Let's Encrypt است:

Bash

sudo apt install certbot python3-certbot-nginx

sudo certbot --nginx -d your\_domain.com -d www.your\_domain.com

1. **به‌روزرسانی بسته‌های سیستم:**
   * به طور منظم بسته‌های سیستم‌عامل و پکیج‌های پایتون را به‌روز کنید.
2. **لاگینگ (Logging):**
   * تنظیمات لاگینگ را در settings.py برای Production به درستی پیکربندی کنید تا خطاها و اطلاعات مهم را در فایل‌ها ذخیره کند.

این سناریو یک دید کلی از مراحل اصلی استقرار Django را ارائه می‌دهد. هر مرحله جزئیات و تنظیمات خاص خود را دارد که باید با توجه به نیازهای پروژه و سرور شما پیکربندی شوند.

فهمیدم! سوال شما این است که چطور یک پروژه جنگو را از محیط توسعه (development) به محیط اجرایی (production) منتقل کنید، **بدون اینکه سورس کدها را مستقیماً روی سرور تولید قرار دهید.** این یک رویکرد امنیتی عالی و منطقی است، به خصوص برای پروژه‌های بزرگ.

در دنیای واقعی، برای این کار از **Docker** و **Containerization** استفاده می‌شود. Docker به شما اجازه می‌دهد پروژه خود را به صورت یک "بسته اجرایی" (Image) درآورید که شامل تمام وابستگی‌ها، کدهای کامپایل شده و تنظیمات لازم است، و سپس آن بسته را روی سرور اجرا کنید، بدون نیاز به انتقال مستقیم سورس کد.

**چرا Docker بهترین راه حل است؟**

* **کپسوله‌سازی (Encapsulation):** تمام وابستگی‌های پروژه (پایتون، جنگو، کتابخانه‌ها، تنظیمات) در یک بسته واحد قرار می‌گیرند.
* **ایزوله‌سازی (Isolation):** پروژه شما از محیط سیستم عامل سرور جدا می‌شود.
* **قابلیت تکرار (Reproducibility):** اطمینان حاصل می‌شود که پروژه شما در هر محیطی دقیقاً یکسان اجرا می‌شود.
* **امنیت:** سورس کد اصلی داخل کانتینر بسته می‌شود و دسترسی مستقیم به آن روی سرور دشوارتر است. تنها چیزی که روی سرور می‌رود، Docker Image کامپایل شده است.
* **انتقال‌پذیری (Portability):** Docker Image را می‌توانید به راحتی بین سرورها منتقل کنید.

**مراحل کلی برای انتقال پروژه جنگو به سرور بدون سورس کد (با استفاده از Docker):**

### 1. پیش‌نیازها

* **Docker Desktop روی ویندوز (محیط توسعه شما):** برای ساخت Docker Image.
* **Docker Engine روی سرور لینوکس (محیط Production شما):** برای اجرای Docker Image.
* **فایل** requirements.txt**:** لیست تمام کتابخانه‌های پایتون که پروژه شما به آن‌ها نیاز دارد. این را با دستور pip freeze > requirements.txt در محیط مجازی پروژه جنگو خود ایجاد کنید.

### 2. آماده‌سازی پروژه برای Docker در PyCharm (در ویندوز)

در ریشه پروژه جنگو خود (کنار manage.py)، فایل‌های زیر را ایجاد کنید:

#### الف) **Dockerfile**

این فایل دستورالعمل ساخت Docker Image را شامل می‌شود.

Dockerfile

# Use an official Python runtime as a parent image

FROM python:3.10-slim-buster

# Set the working directory in the container

WORKDIR /app

# Install any needed packages specified in requirements.txt

# Copy requirements.txt first to leverage Docker cache

COPY requirements.txt /app/

RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code

# IMPORTANT: This copies your source code into the image.

# The idea is that this image is what you "deploy", not the source code itself.

COPY . /app/

# Collect static files (run only once during image build)

# STATIC\_ROOT must be defined in settings.py

# Example: STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

RUN python manage.py collectstatic --noinput

# Expose the port your Django app will run on

EXPOSE 8000

# Run gunicorn (or uWSGI) to serve the Django application

# Replace 'your\_project\_name' with the actual name of your Django project folder

CMD ["gunicorn", "--bind", "0.0.0.0:8000", "your\_project\_name.wsgi:application"]

**توضیحات مهم برای** Dockerfile**:**

* FROM python:3.10-slim-buster: از یک ایمیج پایه پایتون سبک استفاده می‌کند.
* WORKDIR /app: دایرکتوری کاری داخل کانتینر را app/ قرار می‌دهد.
* COPY requirements.txt /app/ **و** RUN pip install ...: ابتدا فقط requirements.txt را کپی و وابستگی‌ها را نصب می‌کند. این کار به Docker اجازه می‌دهد از کش (cache) استفاده کند و در صورت عدم تغییر requirements.txt، مراحل نصب وابستگی‌ها را دوباره انجام ندهد.
* COPY . /app/: **این خط کل سورس کد پروژه شما را در Docker Image کپی می‌کند.** این Docker Image همان "خروجی" شماست که شامل کدهای کامپایل شده (.pyc که پایتون تولید می‌کند)، فایل‌های استاتیک و تمام وابستگی‌ها است.
* RUN python manage.py collectstatic --noinput: فایل‌های استاتیک را داخل ایمیج جمع‌آوری می‌کند. مطمئن شوید که STATIC\_ROOT در settings.py تنظیم شده باشد (مثلاً os.path.join(BASE\_DIR, 'staticfiles')).
* EXPOSE 8000: اعلام می‌کند که کانتینر روی پورت 8000 اجرا می‌شود (می‌توانید تغییر دهید).
* CMD ["gunicorn", ...]: دستوری که کانتینر هنگام شروع اجرا می‌کند. gunicorn یک سرور WSGI است که Django را در محیط Production اجرا می‌کند.
  + your\_project\_name **را با نام واقعی پوشه پروژه جنگو خود (پوشه‌ای که** settings.py **و** wsgi.py **در آن هستند) جایگزین کنید.**
  + **نصب Gunicorn:** باید gunicorn را در requirements.txt اضافه کنید: gunicorn==20.1.0 (یا نسخه دیگر).
  + **پیکربندی** settings.py **برای Production:** حتماً یک فایل settings\_prod.py یا منطقی برای DEBUG=False، ALLOWED\_HOSTS و تنظیمات پایگاه داده Production در settings.py خود داشته باشید.

#### ب) **.dockerignore**

این فایل مانند .gitignore عمل می‌کند و به Docker می‌گوید که چه فایل‌ها و پوشه‌هایی را هنگام ساخت Image نادیده بگیرد. این کار حجم Image را کاهش می‌دهد.

.git

.venv

\_\_pycache\_\_

\*.pyc

\*.sqlite3

.vscode

\*.env

# هر چیز دیگری که نباید در image نهایی باشد

### 3. ساخت Docker Image (در PyCharm در ویندوز)

در ترمینال PyCharm، به ریشه پروژه جنگو خود بروید و دستور زیر را اجرا کنید:

Bash

docker build -t my-django-app:latest .

* my-django-app: نام Image شما.
* latest: تگ Image (می‌تواند شماره نسخه باشد).
* .: به Docker می‌گوید Dockerfile در دایرکتوری فعلی است.

این دستور Docker Image را می‌سازد. تمام کدهای سورس و وابستگی‌ها در این Image بسته‌بندی می‌شوند.

### 4. انتقال Docker Image به سرور (بدون اینترنت)

حالا که Image ساخته شده است، باید آن را به سرور منتقل کنید.

1. **ذخیره Docker Image در یک فایل:**

Bash

docker save -o my-django-app.tar my-django-app:latest

این دستور Image my-django-app:latest را به یک فایل .tar به نام my-django-app.tar ذخیره می‌کند. این فایل شامل تمام لایه‌های Image است.

1. انتقال فایل .tar به سرور:

این فایل my-django-app.tar همان "خروجی بسته بندی شده" پروژه شماست که بدون سورس کد مستقیم قابل اجراست.

* + آن را روی یک **فلش درایو یا هارد اکسترنال** کپی کنید.
  + فلش درایو را به سرور لینوکس متصل کرده و فایل .tar را در یک مسیر مناسب (مثلاً /tmp یا /opt) کپی کنید.

### 5. بارگذاری و اجرای Docker Image در سرور (در لینوکس)

با SSH به سرور لینوکس خود وصل شوید.

1. **بارگذاری Docker Image از فایل:**

Bash

docker load -i /path/to/my-django-app.tar

این دستور Image را از فایل .tar به Docker Engine سرور وارد می‌کند.

1. اجرای کانتینر Django:

حالا می‌توانید یک کانتینر از Image ساخته شده را اجرا کنید:

Bash

docker run -d --name django\_app -p 8000:8000 -v /path/to/media:/app/media\_files -v /path/to/db:/app/db my-django-app:latest

**توضیحات** docker run**:**

* + -d: کانتینر را در پس‌زمینه اجرا می‌کند (detached mode).
  + --name django\_app: یک نام برای کانتینر شما تعیین می‌کند.
  + -p 8000:8000: پورت 8000 داخل کانتینر را به پورت 8000 روی سرور مپ می‌کند.
  + -v /path/to/media:/app/media\_files: **برای فایل‌های مدیا.** این یک Volume مپینگ (volume mapping) است. media\_files/ که در MEDIA\_ROOT خود مشخص کرده‌اید (/app/media\_files در کانتینر) را به یک پوشه دائمی روی سیستم فایل سرور (/path/to/media) وصل می‌کند. این برای این است که فایل‌های آپلود شده (مدیا) حتی با حذف یا به‌روزرسانی کانتینر، باقی بمانند. **مسیرهای واقعی خود را جایگزین کنید.**
  + -v /path/to/db:/app/db: **برای پایگاه داده SQLite (اگر استفاده می‌کنید).** اگر از SQLite استفاده می‌کنید و فایل db.sqlite3 شما در ریشه پروژه است، این خط را اضافه کنید تا فایل دیتابیس هم دائمی باشد. (/app/db را به جایی که فایل db.sqlite3 در کانتینر قرار دارد تغییر دهید). **اگر از PostgreSQL/MySQL جداگانه استفاده می‌کنید، این خط را حذف کنید.**
  + my-django-app:latest: نام و تگ Image که باید اجرا شود.

### 6. پیکربندی Nginx (روی سرور)

حالا Nginx روی سرور شما باید درخواست‌ها را به کانتینر Django (که روی پورت 8000 سرور شما در حال اجراست) پروکسی کند و فایل‌های استاتیک و مدیا را مستقیماً Serve کند.

Nginx

server {

listen 80;

server\_name your\_domain.com your\_server\_ip; # دامنه یا IP سرور شما

# Serve static files directly from the host (outside the container)

# This path should point to the 'staticfiles' directory created by collectstatic

location /static/ {

alias /path/to/your/django/project/staticfiles/; # مسیر واقعی collected\_static\_files روی سرور

expires 30d; # کش کردن فایل‌های استاتیک برای 30 روز

}

# Serve media files directly from the host (outside the container)

# This path should point to the 'media\_files' directory (the Docker volume path)

location /media/ {

alias /path/to/media/; # مسیر واقعی media\_files روی سرور که به کانتینر مپ شده

expires 7d; # کش کردن فایل‌های مدیا

}

# Proxy requests to the Django container

location / {

proxy\_pass http://127.0.0.1:8000; # پورت 8000 روی لوکال هاست سرور (که Docker به آن متصل است)

proxy\_set\_header Host $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;

}

}

**توضیحات Nginx:**

* location /static/ و location /media/: این‌ها باید به مسیرهای واقعی روی فایل‌سیستم سرور شما (که خارج از کانتینر هستند) اشاره کنند. staticfiles با collectstatic داخل کانتینر ایجاد شد و شما باید آن را به بیرون کپی کنید (یا STATIC\_ROOT را در یک Volume دائمی قرار دهید). media\_files همان Volue‌ای است که در docker run مپ کردید.
* proxy\_pass http://127.0.0.1:8000;: Nginx درخواست‌ها را به پورت 8000 روی localhost سرور شما ارسال می‌کند. Docker این پورت را به پورت 8000 کانتینر Django شما هدایت می‌کند.

### خلاصه جریان کار:

1. **در ویندوز (PyCharm):**
   * requirements.txt را ایجاد کنید.
   * Dockerfile و .dockerignore را بنویسید.
   * docker build را اجرا کنید تا Docker Image ساخته شود.
   * docker save را اجرا کنید تا Image به یک فایل .tar تبدیل شود.
2. **انتقال آفلاین:**
   * فایل .tar را روی فلش درایو کپی کنید.
   * فلش درایو را به سرور لینوکس منتقل کنید.
3. **در سرور لینوکس:**
   * Docker Engine را نصب کنید.
   * docker load را اجرا کنید تا Image وارد Docker شود.
   * docker run را اجرا کنید تا کانتینر Django شروع به کار کند.
   * **Nginx** را برای Serve کردن استاتیک/مدیا و پروکسی کردن به کانتینر Django پیکربندی کنید.
   * **فایروال** را برای باز کردن پورت 80 (HTTP) یا 443 (HTTPS) پیکربندی کنید.

این روش "سورس کد" را به معنای فایل‌های خام .py مستقیماً روی سرور Production قرار نمی‌دهد، بلکه آن‌ها را در قالب یک Docker Image (که حاوی بایت‌کدها و تمام چیزهای لازم است) بسته‌بندی می‌کند. این حرفه‌ای‌ترین و امن‌ترین روش برای دیپلوی پروژه‌های بزرگ جنگو است.

بله، سوالات بسیار دقیق و مهمی هستند! دقیقاً همینجاست که قدرت و مزیت اصلی داکر (Docker) خودش رو نشون می‌ده.

**۱. آیا با داکر کردن پروژه لازم نیست نرم‌افزار دیگه ای نصب بشه؟**

**تقریباً بله، اما با یک توضیح مهم:**

* **روی سیستم عامل سرور (Host OS):**
  + **تنها چیزی که حتماً باید نصب باشه، خود** Docker Engine **هست.** داکر بقیه کارها رو انجام می‌ده.
  + **وب‌سرور (مثل Nginx یا Apache) همچنان باید نصب باشه.** چرا؟ چون Nginx/Apache وظیفه اصلی دریافت درخواست‌های ورودی از اینترنت (پورت 80 و 443) و مدیریت فایل‌های استاتیک و مدیا رو به عهده داره. داکر اپلیکیشن جنگو شما رو اجرا می‌کنه، اما Nginx بین اینترنت و داکر شما یک لایه واسط است. (مگر اینکه تصمیم بگیرید اپلیکیشن داکر رو مستقیماً به اینترنت وصل کنید که اصلاً توصیه نمیشه).
  + **پایگاه داده (مثلاً PostgreSQL, MySQL):** اگر از پایگاه داده خارجی (غیر از SQLite که داخل کانتینر میتونه باشه) استفاده می‌کنید، نرم‌افزار پایگاه داده (مثل MySQL Server) باید روی سرور یا یک سرور جداگانه نصب و در حال اجرا باشه. داکر فقط اپلیکیشن جنگو رو اجرا می‌کنه، نه دیتابیس رو (مگر اینکه دیتابیس رو هم داخل یک کانتینر داکر دیگه اجرا کنید که خودش یک موضوع جداگانه است).
* **داخل کانتینر داکر:**
  + **نه!** وقتی شما Docker Image رو می‌سازید (با دستور docker build)، **تمام نیازمندی‌های پروژه جنگوی شما (پایتون، جنگو، Gunicorn، تمام کتابخانه‌های** requirements.txt **و حتی فایل‌های استاتیک جمع‌آوری شده)** داخل همون Image بسته‌بندی می‌شن.
  + به همین دلیله که میگیم داکر، پروژه رو "خودبسنده" (self-contained) می‌کنه. کانتینر داکر یک محیط ایزوله هست که همه چیز لازم رو برای اجرا با خودش داره.

**خلاصه:** روی سرور، شما به Docker Engine و یک وب‌سرور (مثل Nginx) نیاز دارید. اگر دیتابیس خارجی دارید، به سرور دیتابیس هم نیاز دارید. اما داخل کانتینر داکر، هیچ نرم‌افزار اضافی نصب نمیشه، چون همه چیز از قبل توی Image هست.

**۲. داکر نیازمندی‌ها را با خودش داره؟**

**بله، دقیقاً!** این یکی از اصلی‌ترین مزایای داکر هست.

وقتی شما Dockerfile رو می‌نویسید و دستور docker build رو اجرا می‌کنید:

* FROM python:3.10-slim-buster: این خط به داکر میگه که از یک سیستم عامل پایه (مثل Debian) و یک نسخه مشخص از پایتون شروع کنه. این پایتون و ابزارهای سیستم عامل پایه (مثل apt برای نصب پکیج‌ها) در Image قرار می‌گیرند.
* RUN pip install -r requirements.txt: این دستور تمام کتابخانه‌های پایتون که در requirements.txt لیست کردید (مثل جنگو، Gunicorn، Pillow و...) رو دانلود و نصب می‌کنه. این کتابخانه‌ها هم داخل Image قرار می‌گیرند.
* COPY . /app/: این خط کد سورس پروژه شما رو کپی می‌کنه.
* RUN python manage.py collectstatic: این فایل‌های استاتیک پروژه رو جمع‌آوری می‌کنه و اون‌ها هم داخل Image قرار می‌گیرند.

بنابراین، **Docker Image نهایی یک بسته کامل هست که هرچیزی که پروژه جنگو برای اجرا نیاز داره رو با خودش حمل می‌کنه.** شما این Image رو به سرور منتقل می‌کنید و Docker Engine روی سرور اون رو اجرا می‌کنه. دیگه نیازی نیست روی سرور pip install کنید یا پایتون رو نصب کنید (مگر اینکه برای کارهای دیگه نیاز داشته باشید، ولی برای اجرای این کانتینر نه).

**۳. داخل داکر دسترسی چطور میشه؟ کاربر دیگری می‌تونه به داخل داکر دسترسی داشته باشه؟ و محتویات را ببینه؟**

این موضوع مربوط به امنیت کانتینرها و نحوه دسترسی به اون‌هاست:

* **دسترسی به داخل کانتینر (Host OS به Container):**
  + به صورت پیش‌فرض، کانتینرها ایزوله هستند. شما به عنوان ادمین سرور، می‌تونید با دستور docker exec وارد یک کانتینر در حال اجرا بشید و به فایل‌ها و محیط داخل اون دسترسی پیدا کنید:

Bash

docker exec -it my\_django\_production\_app bash

با این دستور، شما یک شل (shell) در داخل کانتینر دریافت می‌کنید و می‌تونید فایل‌ها رو ببینید، دستورات رو اجرا کنید و...

* + **پس بله، اگر کسی دسترسی SSH به سرور لینوکس شما (همون Host OS) رو داشته باشه و Docker Engine روی سرور نصب باشه، می‌تونه وارد کانتینر بشه و به محتویات اون دسترسی پیدا کنه.**
* **دیدن محتویات (سورس کد) داخل کانتینر:**
  + **فایل‌های سورس کد پایتون (**.py**): بله، قابل مشاهده هستند.** وقتی شما COPY . /app/ رو در Dockerfile میذارید، فایل‌های .py (سورس کد) هم داخل Image کپی میشن.
  + **فایل‌های** .pyc **(بایت‌کد کامپایل شده):** این فایل‌ها هم داخل کانتینر هستند. این فایل‌ها بایت‌کد هستند و خواندن مستقیمشون سخته، اما با ابزارهای decompiler قابل تبدیل به کد پایتون (تقریباً مشابه سورس اصلی) هستند.
  + **نتیجه:** داکر امنیت رو در لایه "انتقال سورس کد به سرور" بالا می‌بره (یعنی شما فایل‌های خام رو کپی نمی‌کنید)، اما **داکر یک ابزار رمزنگاری یا مبهم‌سازی کد نیست.** اگر یک نفر به سرور شما دسترسی ادمین پیدا کنه، میتونه به کانتینر شما دسترسی پیدا کنه و سورس کدهایی که داخل Image کپی کردید رو ببینه.
* **امنیت سورس کد در داکر:**
  + **داکر برای "پنهان کردن" سورس کد از ادمین سرور طراحی نشده.** وظیفه اصلی داکر ایزوله‌سازی محیط و تسهیل دیپلوی هست.
  + برای حفاظت از سورس کد در برابر ادمین سرور (که به docker exec دسترسی داره)، راه‌حل‌های پیچیده‌تر و خاصی (مثل Code Obfuscation در پایتون، که قبلاً توضیح دادم و معمولاً توصیه نمیشه) وجود داره که هیچ‌کدوم ۱۰۰٪ تضمینی نیستند.
  + **راهکار عملی‌تر:** امنیت سرور فیزیکی/مجازی، کنترل دسترسی‌ها (چه کسی به SSH دسترسی داره)، استفاده از ابزارهای مانیتورینگ و امنیتی، و استفاده از ریپازیتوری‌های کد امن (مثل Git خصوصی).

خلاصه پاسخ به سوال دسترسی:

اگر کسی به سرور شما دسترسی ادمین داشته باشه (یعنی بتونه SSH بزنه و دستورات sudo و docker رو اجرا کنه)، می‌تونه وارد کانتینر شما بشه و سورس کدهایی که داخل Image کپی کردید رو ببینه. داکر از این سطح دسترسی محافظت نمی‌کنه. داکر از بهم ریختگی وابستگی‌ها و محیط، و از نیاز به نصب دستی نرم‌افزارها روی سرور جلوگیری می‌کنه.

امیدوارم این توضیحات کامل و روشن بوده باشه. این سوالات نشان‌دهنده درک عمیق شما از معماری سیستم هستن!

سوال خوبی پرسیدید. بیایید دقیقاً به بحث رمزگذاری روی داکر و امنیت آن بپردازیم.

**۱. آیا میشه روی داکر "رمز" گذاشت؟**

به طور مستقیم، **شما نمی‌توانید روی یک Docker Image یا یک Docker Container در حال اجرا، یک "رمز عبور" به معنای رمز عبور فایل‌های فشرده (مثل ZIP یا RAR) بگذارید که کاربر برای ورود به آن کانتینر یا دیدن محتویات Image، نیاز به وارد کردن آن رمز داشته باشد.**

داکر برای این نوع رمزگذاری و کنترل دسترسی در سطح محتویات Image/Container طراحی نشده است. دسترسی به محتویات Image و Container در حال اجرا، تابعی از **دسترسی به Docker Daemon (موتور داکر) و دسترسی به سیستم عامل هاست (سرور لینوکس)** است.

**۲. آیا امنیت رمزگذاری برای داکر تامین شده؟ یا این هم باز میشه؟**

همانطور که در پاسخ قبلی اشاره شد، داکر به خودی خود ابزاری برای رمزنگاری یا مبهم‌سازی سورس کد نیست. امنیت در داکر و کانتینرها در لایه‌های مختلفی تامین می‌شود، اما نه از طریق "رمز عبور مستقیم روی کانتینر":

**الف) امنیت در سطح Image (ایمیج داکر):**

* **ذخیره Image در فایل (**docker save**):** وقتی شما Image را با docker save به یک فایل .tar تبدیل می‌کنید، می‌توانید آن فایل .tar را با ابزارهای سیستم عامل (مثل 7-Zip در ویندوز یا gpg در لینوکس) **رمزنگاری کنید**.
  + **روش کار:** قبل از انتقال، فایل my-django-project.tar را با رمز عبور فشرده (مثلاً با 7-Zip و رمزگذاری AES-256) یا با ابزارهای رمزنگاری فایل (gpg) رمزگذاری کنید.
  + **امنیت:** این روش، **امنیت فایل را در حین انتقال فیزیکی (مثلاً روی فلش درایو) تضمین می‌کند.** یعنی اگر فایل به دست یک فرد غیرمجاز بیفتد، بدون رمز عبور نمی‌تواند آن را باز و محتویاتش را ببیند.
  + **آیا باز می‌شود؟** بله، با داشتن رمز عبور صحیح و ابزار رمزگشایی مربوطه، فایل باز می‌شود و Image از آن استخراج می‌شود.
* **ذخیره Image در Docker Registry (رجیستری داکر):**
  + اگر قرار است Image را از طریق شبکه منتقل کنید (مثلاً به یک سرور دیگر با دسترسی اینترنت)، بهترین راه استفاده از یک **Docker Registry خصوصی و امن** است (مثل Docker Hub Private Repositories, GitLab Container Registry, AWS ECR و...).
  + **امنیت:** این رجیستری‌ها معمولاً با **احراز هویت (Authentication)** و **مجوز دسترسی (Authorization)** محافظت می‌شوند. فقط کاربرانی که credential (نام کاربری و رمز عبور) صحیح دارند، می‌توانند Imageها را Push (آپلود) یا Pull (دانلود) کنند. ارتباط هم معمولاً از طریق HTTPS رمزگذاری شده است.
  + **آیا باز می‌شود؟** Image به صورت عادی Pull می‌شود و روی سیستم مقصد قرار می‌گیرد. هدف اینجا امنیت در انتقال و کنترل دسترسی به Imageها در یک مخزن مرکزی است.

**ب) امنیت در سطح Container (کانتینر داکر در حال اجرا):**

هنگامی که Image روی سرور بارگذاری و به عنوان یک کانتینر اجرا می‌شود، موضوع "رمز" به معنای بالا بی‌معنی می‌شود. امنیت کانتینر به عوامل زیر بستگی دارد:

1. **امنیت سیستم عامل هاست (سرور لینوکس):** این مهمترین لایه است. اگر سرور لینوکس شما هک شود یا کسی دسترسی روت/sudo پیدا کند، می‌تواند:
   * به Docker Daemon دسترسی پیدا کند.
   * دستور docker exec را برای ورود به کانتینر اجرا کند.
   * فایل‌های داخل کانتینر را از سیستم فایل هاست (در صورت استفاده از Storage Driverهای خاص) ببیند.
   * **پس، اگر سیستم عامل هاست امن نباشد، هیچ "رمزی" روی کانتینر نمی‌تواند از محتویات آن محافظت کند.**
2. **کنترل دسترسی به Docker Daemon:** فقط کاربران مجاز (که عضو گروه docker هستند یا دسترسی sudo دارند) باید بتوانند دستورات docker را اجرا کنند. این جلوگیری می‌کند از اینکه کاربران عادی روی سرور شما، بتوانند کانتینرها را بررسی یا دستکاری کنند.
3. **Permissions (مجوزهای فایل) داخل کانتینر:**
   * کدهای شما داخل کانتینر با یک کاربر خاص (مثلاً root یا یک کاربر غیر root که در Dockerfile تعریف می‌کنید) اجرا می‌شوند.
   * می‌توانید و باید مطمئن شوید که فایل‌های سورس کد شما در داخل Image (و در نتیجه کانتینر) فقط برای کاربر اجرای اپلیکیشن قابل خواندن باشند و نه برای سایر کاربران فرضی داخل کانتینر (اگرچه در یک کانتینر معمولاً یک اپلیکیشن اجرا می‌شود).
4. **Content Trust (اعتماد به محتوا):**
   * Docker Content Trust به شما اجازه می‌دهد Imageها را با کلیدهای رمزنگاری امضا کنید. این کار تضمین می‌کند که Imageای که Pull می‌کنید، واقعاً همان Imageای است که توسط ناشر مورد نظر ساخته شده و دستکاری نشده است. این بیشتر برای جلوگیری از دستکاری ناخواسته یا مخرب Imageها در طول مسیر انتقال است، نه پنهان کردن محتوا.

**نتیجه‌گیری نهایی در مورد "رمز" روی داکر:**

* **برای امنیت در حین انتقال آفلاین:** فایل .tar حاصل از docker save را با نرم‌افزارهای فشرده‌سازی دارای رمز (مثل 7-Zip) رمزگذاری کنید. این سطح از رمزگذاری از محتویات در برابر دسترسی در زمان **انتقال فیزیکی** محافظت می‌کند.
* **برای امنیت سورس کد در سرور Production:** داکر به طور مستقیم "رمز"ی برای ورود به کانتینر یا پنهان کردن سورس کد در برابر ادمین سرور ارائه نمی‌دهد. اگر یک ادمین مخرب به سرور شما دسترسی پیدا کند، می‌تواند به داخل کانتینر شما برود و سورس کدهایی که با COPY . /app/ در Image قرار داده‌اید را ببیند.
  + **راهکار اصلی برای حفاظت از سورس کد در سرور، اعتماد به امنیت سرور و کنترل دقیق دسترسی‌های SSH است.** اگر نگران دسترسی ادمین سرور هستید، راه‌حل‌ها پیچیده‌تر می‌شوند (مثلاً اجرای داکر روی ماشین‌های مجازی رمزنگاری شده، یا استفاده از مبهم‌سازی کد پایتون که قبلاً به معایبش اشاره شد). اما این سناریو برای اکثر کاربران عادی و حتی شرکت‌ها رایج نیست و داکر برای حل آن طراحی نشده.

به زبان ساده، داکر کد شما را "بسته‌بندی" می‌کند تا راحت‌تر جابجا شود و اجرا شود، نه اینکه آن را "پنهان" کند. برای پنهان‌سازی واقعی، باید از روش‌های رمزنگاری یا ابهام‌سازی کد در لایه‌های بالاتر استفاده کرد.