# 第四章 雲端、虛擬化、自動化整合

## 4.4 Dockerfile與Docker compose技術

### 4.4.1 客戶需求

開發團隊想要架設網站服務，希望服務能夠快速啟動且不占太多空間，另外因為各個同仁使用的系統和負責的部分不同，希望可以保持開發環境一致且各個服務不會互相干擾。

### 4.4.2 解決方案

一般在建立網站時會需要網頁伺服器、資料庫、框架等服務，我們可以使用Docker容器的技術將每一種服務各自包進一個容器裡面，利用Dockerfile創造客製化的映像檔建構合適的容器，再利用docker compose的技術設定容器的連接並在設定完成後快速啟動所有服務。

### 4.4.3 佈署架構

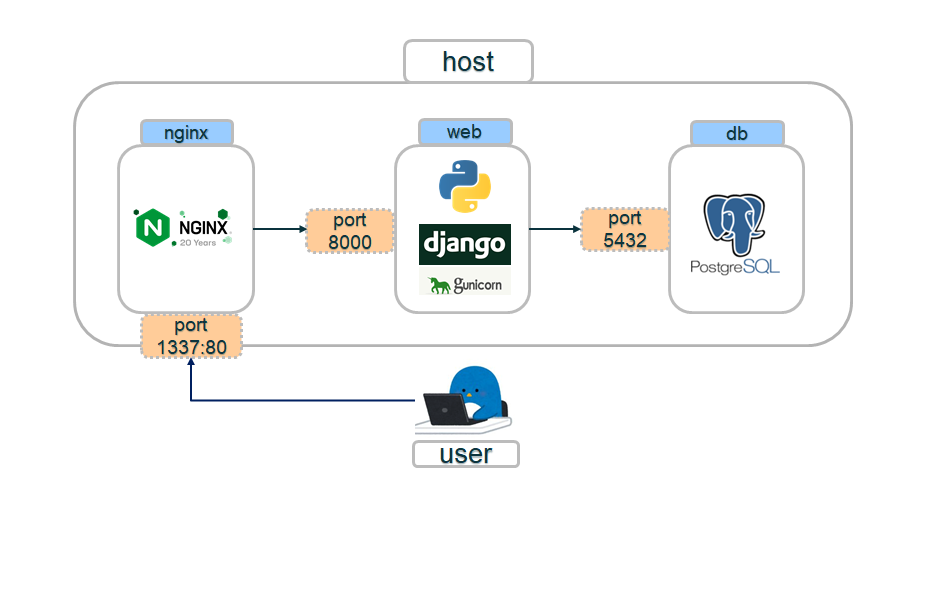


圖4-1 DAG架構圖

在本次的專題中，架設了三個容器。分別是db、web、nginx。

db容器是處理資料庫的容器，這邊選擇使用PostgreSQL並開啟port5432聆聽。

web容器是處理網頁服務的容器，使用基於python語言的網路框架Django和Gunicorn作為WSGI伺服器轉發網頁伺服器的訊息至django和接收django訊息回覆至網頁伺服器，本容器開啟port 8000聆聽。

nginx容器是作為網頁伺服器用，在此使用Nginx作為網頁伺服器，執行反向代理處理用戶端的請求和結果的回覆，對外的port從80改為1337。

### 4.4.4技術說明

Docker是一個開源的容器化平台，可快速地建置、測試和部署應用程式。開發人員可以利用docker的容器化技術將應用程式的程式碼及執行時所需的項目打包在一起，形成一個獨立、一致的運行容器，快速的運行在不同環境裡。

容器的建立需要透過映像檔(docker image)，在官方的平台上有許多映像檔可以直接拉下來用，但有時可能不太符合我們的需求，這時我們可以使用Dockerfile，Dockerfile是一個純文字檔案，裡面包含了所有建構容器環境的指令，可以透過撰寫Dockerfile產生客製化的映像檔。在本次專案中nginx容器和web容器是使用dockerfile設定所需要的環境。

Docker Compose是一個設定和管理多種容器的管理工具。開發者可將容器的所有資訊寫進docker-compose.yml檔裡，設定各個容器的網路、端口配置、開啟順序、資料位置等等，最後再透過docker-compose up指令一次啟動運行所有容器。

### 4.4.5實作建置

首先，建立Dockerfile來設定Web容器的環境，在此為了要讓應用程式的效率更好且更輕薄，我使用 multi-stage builds的方式在dockerfile的前半部分建立builder，使用了python:3.11.4-slim-buster作為基礎，安裝了編譯器gcc和偵錯程式flake8並去掉不相關的部分，接著使用wheel包裝在requirements.txt裡面的服務Django(網頁框架)、psycopg2-binary(用來連接postgreSQL)、gunicorn(WSGI伺服器，處理nginx和Django之間的流量)並指定好版本。

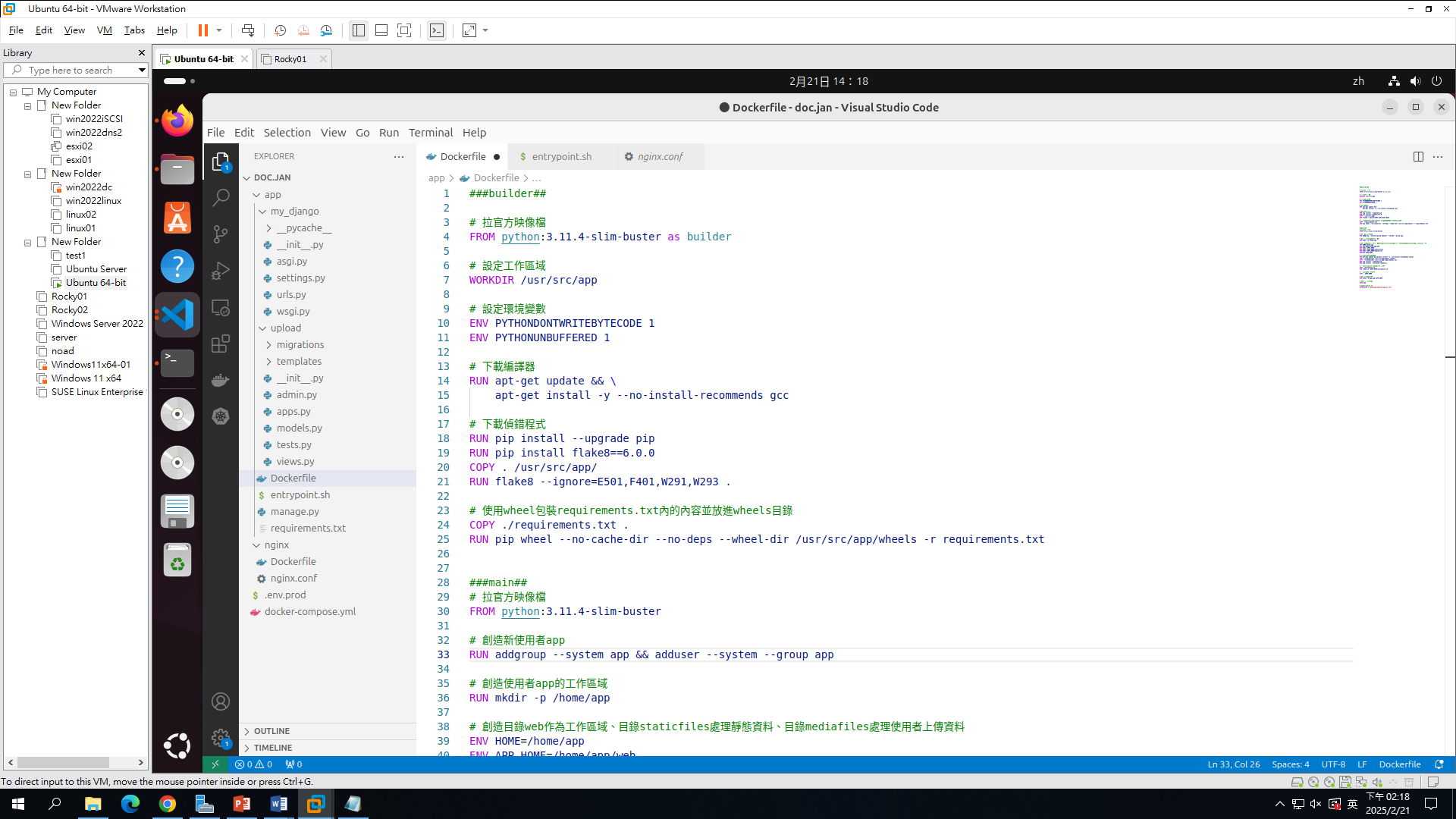


圖4-2 Dockerfile建立builder

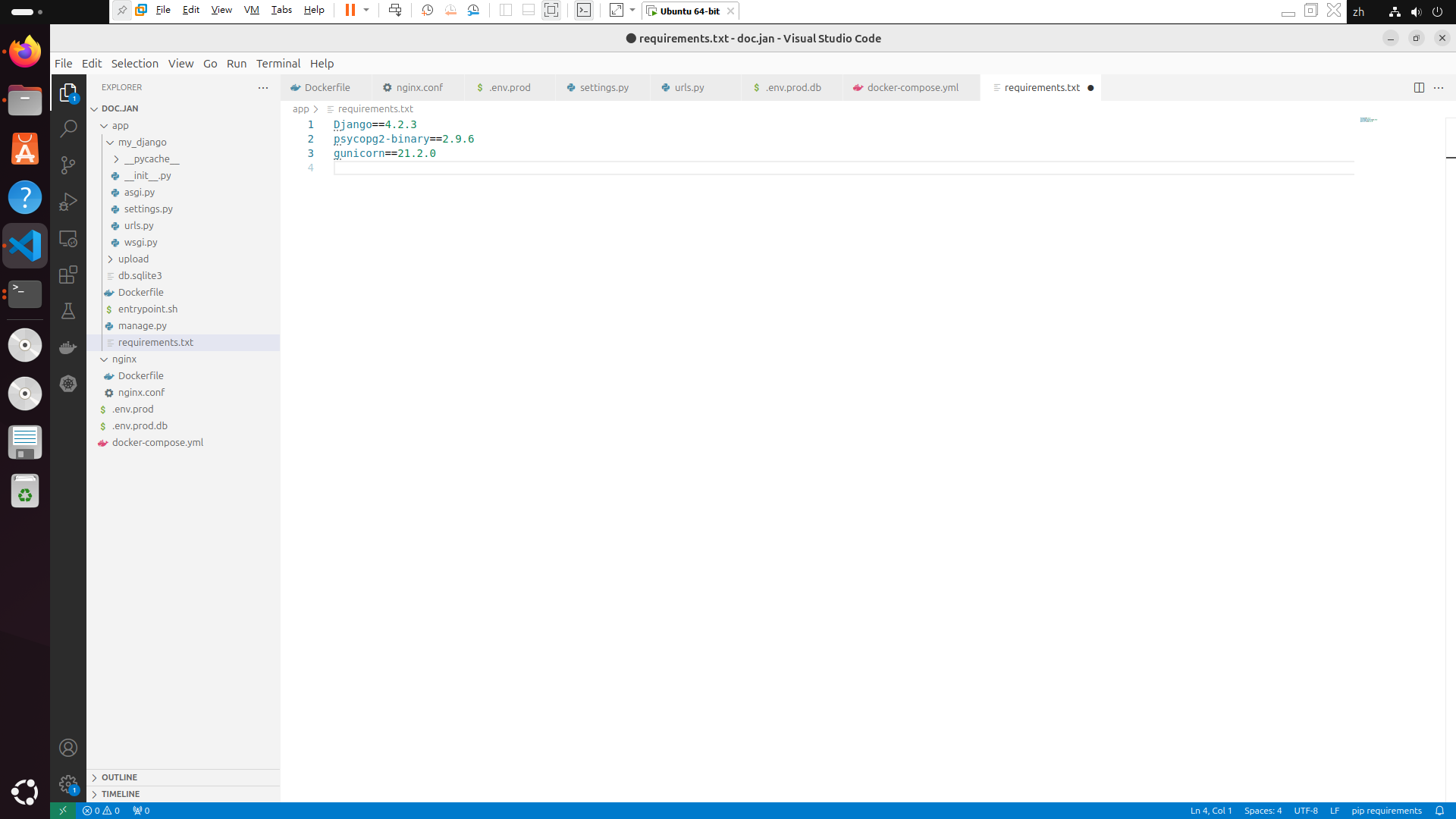


圖4-3 requirements.txt

接著是web容器的主要部分，這邊使用python:3.11.4-slim-buster作為基礎，設定了新的使用者app和需要的目錄、安裝builder的內容和跑entrypoint.sh的內容，在這邊特別創造新的使用者避免駭客入侵時可以直接使用root帳號進行破壞。

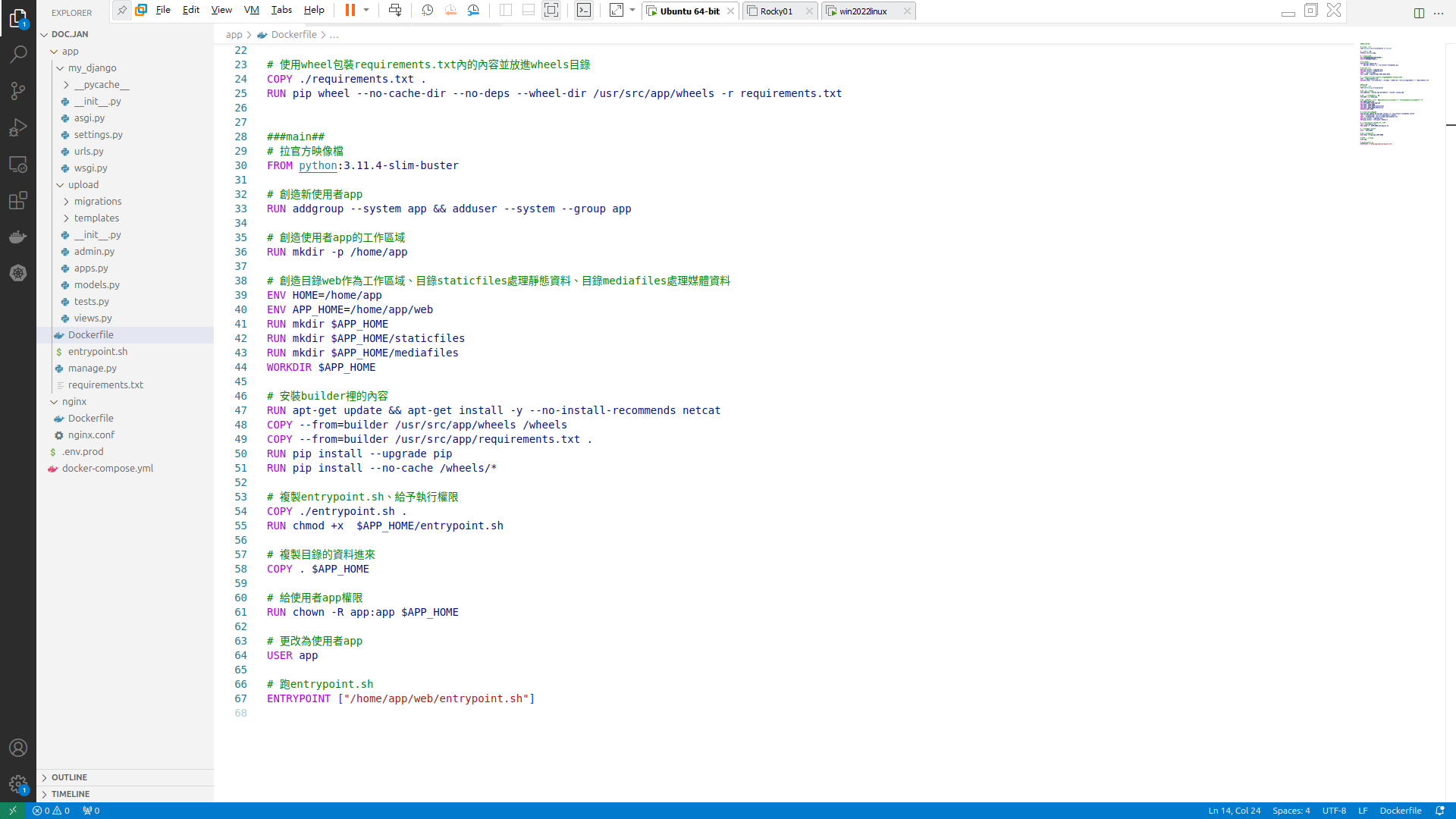


圖4-4 Dockerfile主要部分

在enterpoint.sh這裡，設定了先等待資料庫，在和資料庫連線後同步資料和收集靜態文件。

一張含有 文字, 螢幕擷取畫面, 軟體, 電腦圖示 的圖片

AI 產生的內容可能不正確。

圖4-5 entrypoint.sh

接著，在nginx的部分因為需要修改設定檔，所以也寫了Dockerfile來刪掉預設檔案換成自己修改的設定檔conf.d

一張含有 文字, 螢幕擷取畫面, 軟體, 電腦圖示 的圖片

AI 產生的內容可能不正確。

圖4-6 nginx的Dockerfile

conf.d的部分，設定讓導向web容器的流量轉為導向nginx，發揮代理伺服器的功能。

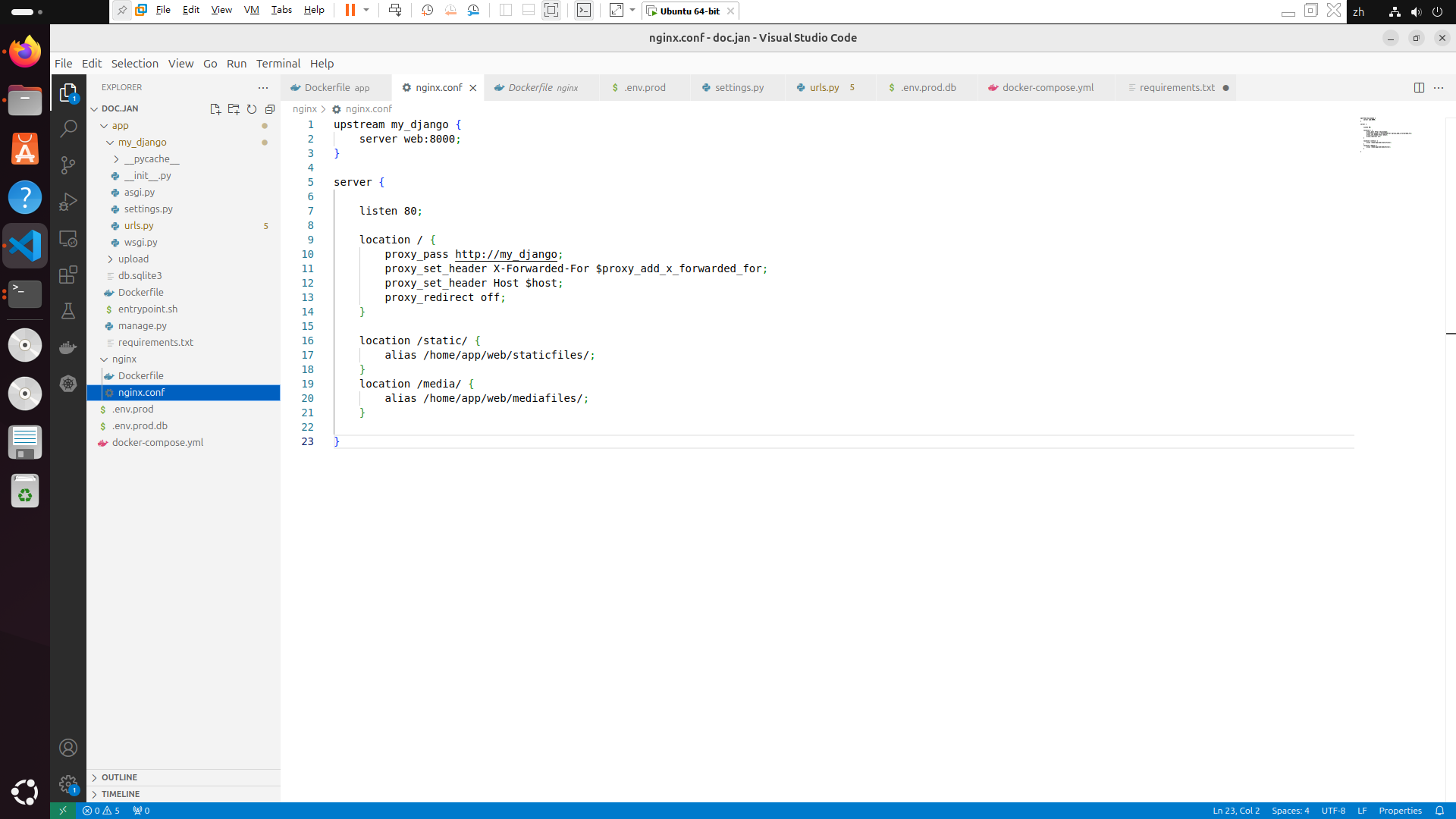


圖4-7 nginx的設定檔conf.d

接著，在nginx的部分因為需要修改設定檔，所以也寫了Dockerfile來刪掉預設檔案換成自己修改的設定檔conf.d

最後是docker-compose.yml的部分，這邊在services建立了三個容器，分別是web、db、nginx。

在web的容器裡設定使用app目錄裡的Dockerfile產生映像檔、使用WSGI伺服器gunicorn、存靜態檔案和媒體檔案的檔案路徑、對外port 8000、環境變數和設定等待db容器後再開始啟動。

在db容器裡則是拉官方映像檔posetgres:15，設定資料庫儲存的位置和環境變數。

在nginx容器裡設定使用nginx資料夾的Dockerfile生成映像檔，接著設定靜態檔案和多媒體檔案的檔案路徑和將對外的port從80改為1337，最後再設定等待web容器啟動後再啟動nginx容器。

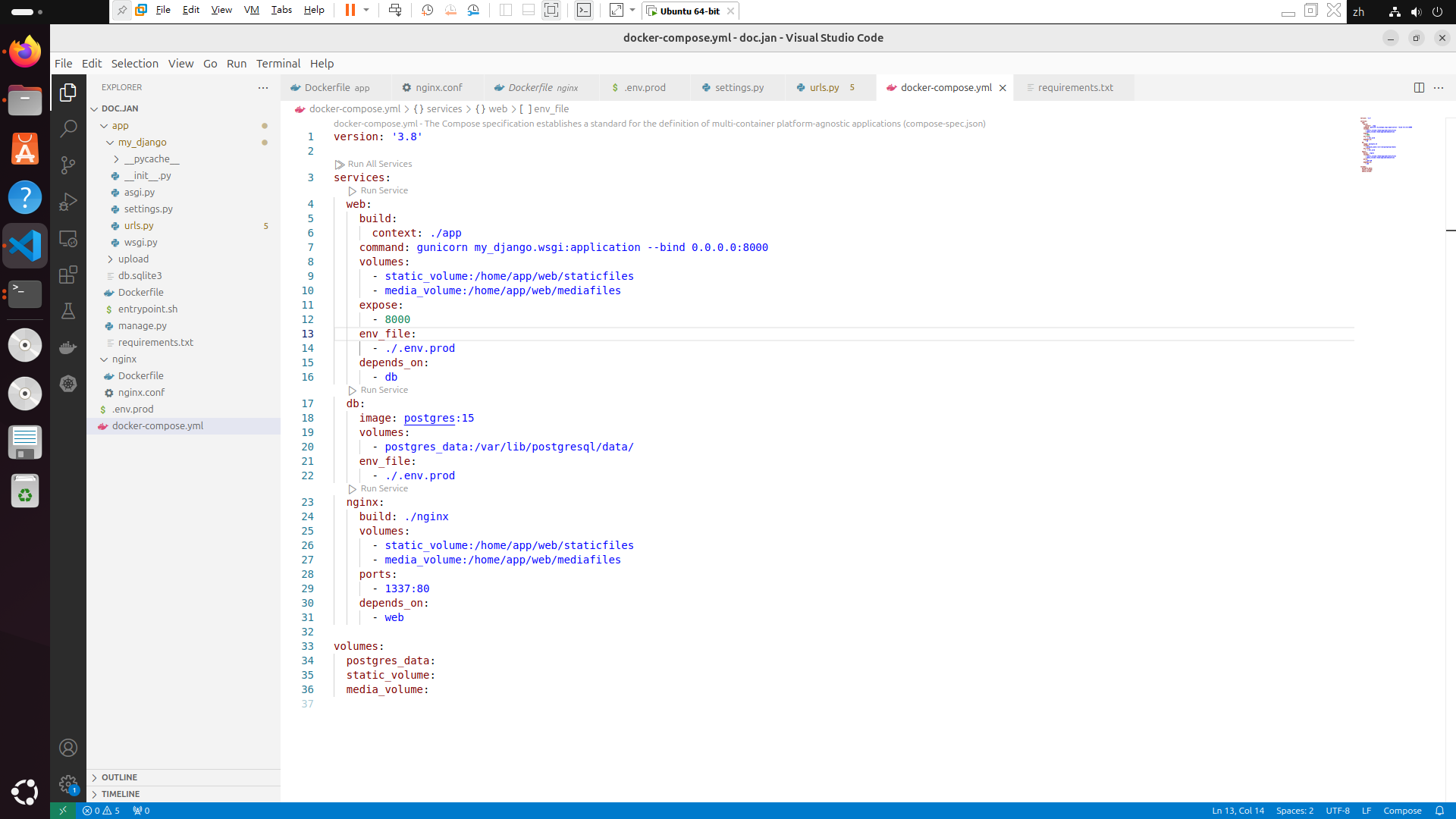


圖4-8 docker-compose.yml

另外，在web和db的環境變數設定檔案如下

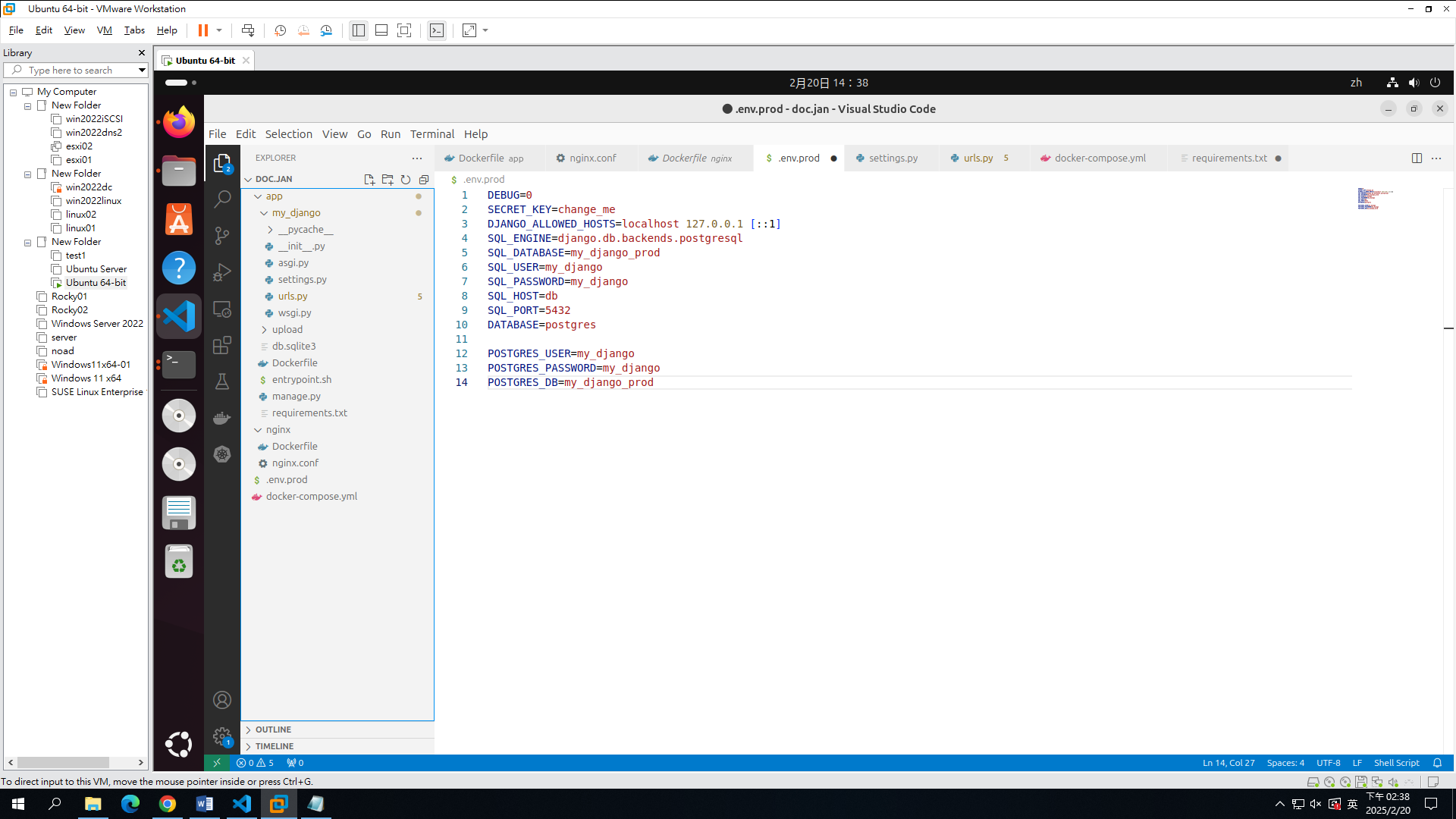


圖4-9 .env.prod

這邊為了方便查看和修改便將django裡setting.py的部分參數拉出來，另外也擔心修改其中一邊的使用者名稱、密碼或資料庫名字後另一邊忘記修改，所以也把db的環境設定放進來。

以下是整體目錄的架構，其中裡面my\_django、upload的資料夾和manage.py檔案是Django的資料，服務內容為讓網站可以執行後台登入和上傳媒體資料，接下來啟動之後會測試這兩樣功能是否能正常運行。

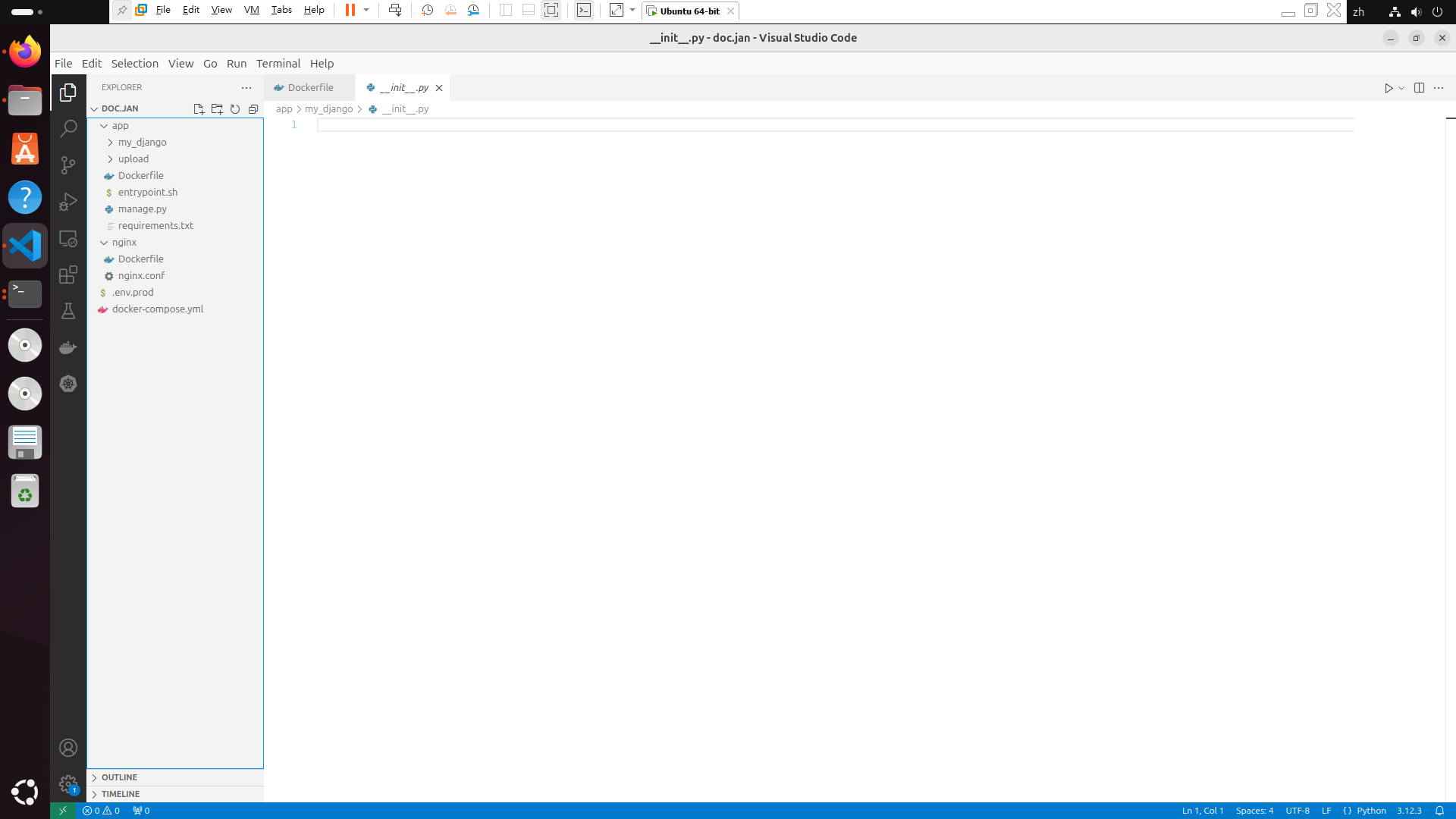
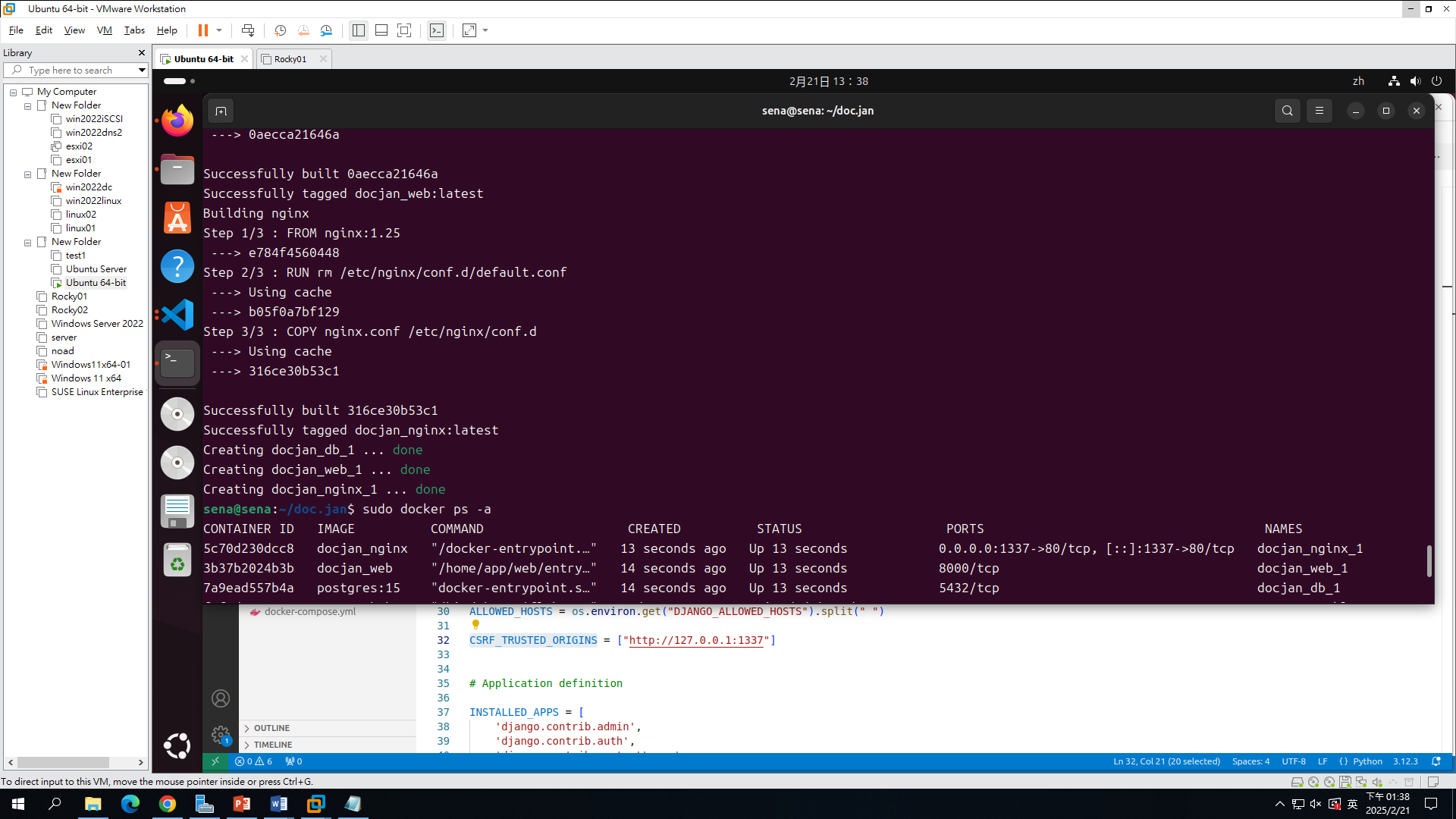


圖4-10 整體目錄

最後來進行測試，首先，使用docker comopose up指令啟動容器，可看到三個容器皆啟動完成，再由docker ps指令確認有開啟正確的port。



三個容區port設置正確

三個容區皆啟動完成

圖4-11 容區啟動資料

接著來進行後台登入功能測試，圖4-12中可看到登入畫面有成功起動，圖4-13可看到使用者後登入成功的畫面。

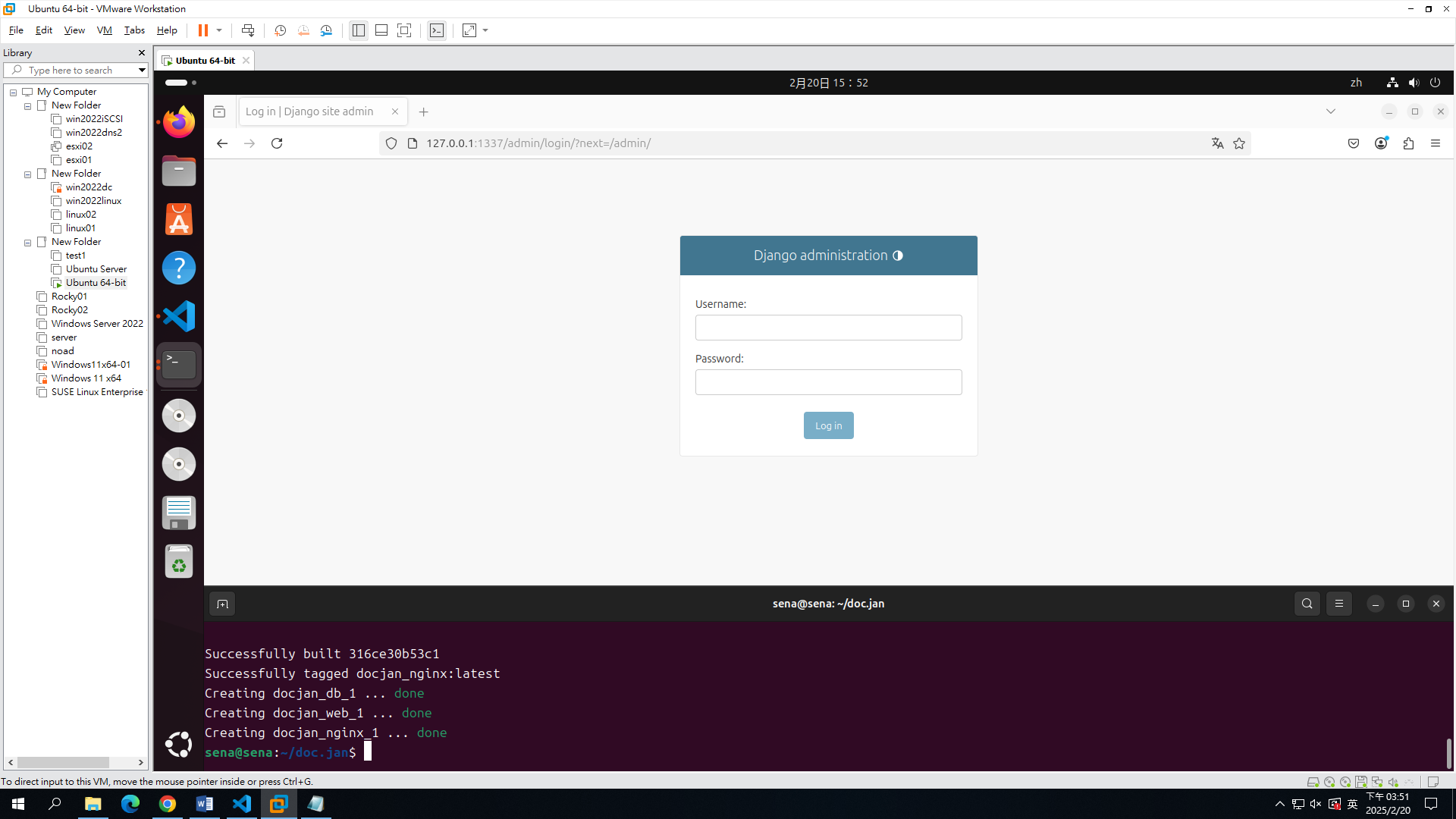


圖4-12 後台登入畫面

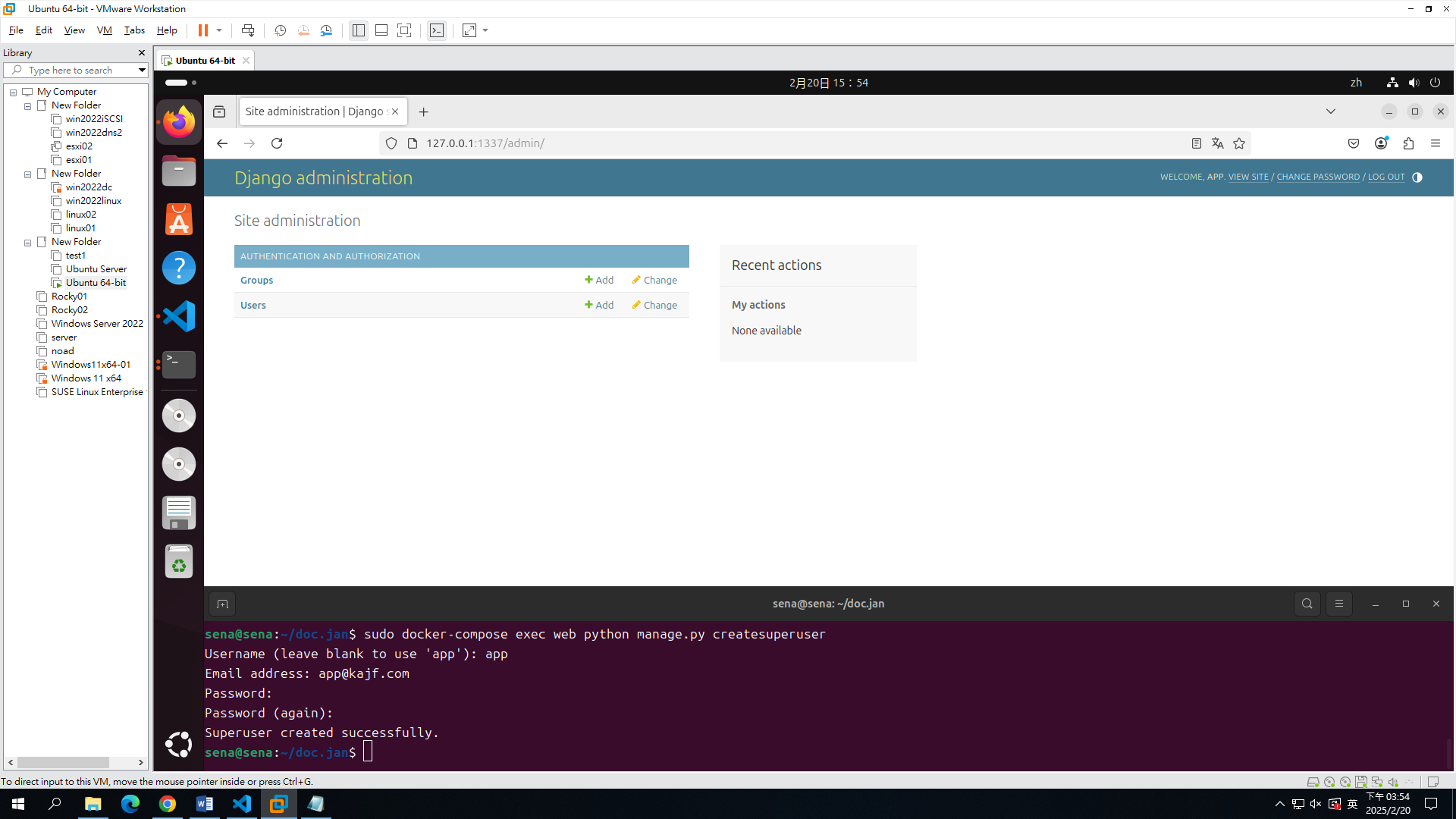


圖4-13 登入成功

接下來是測試上傳媒體檔案功能，可以從圖4-14、圖4-15和圖4-16看到上傳畫面成功並顯示。

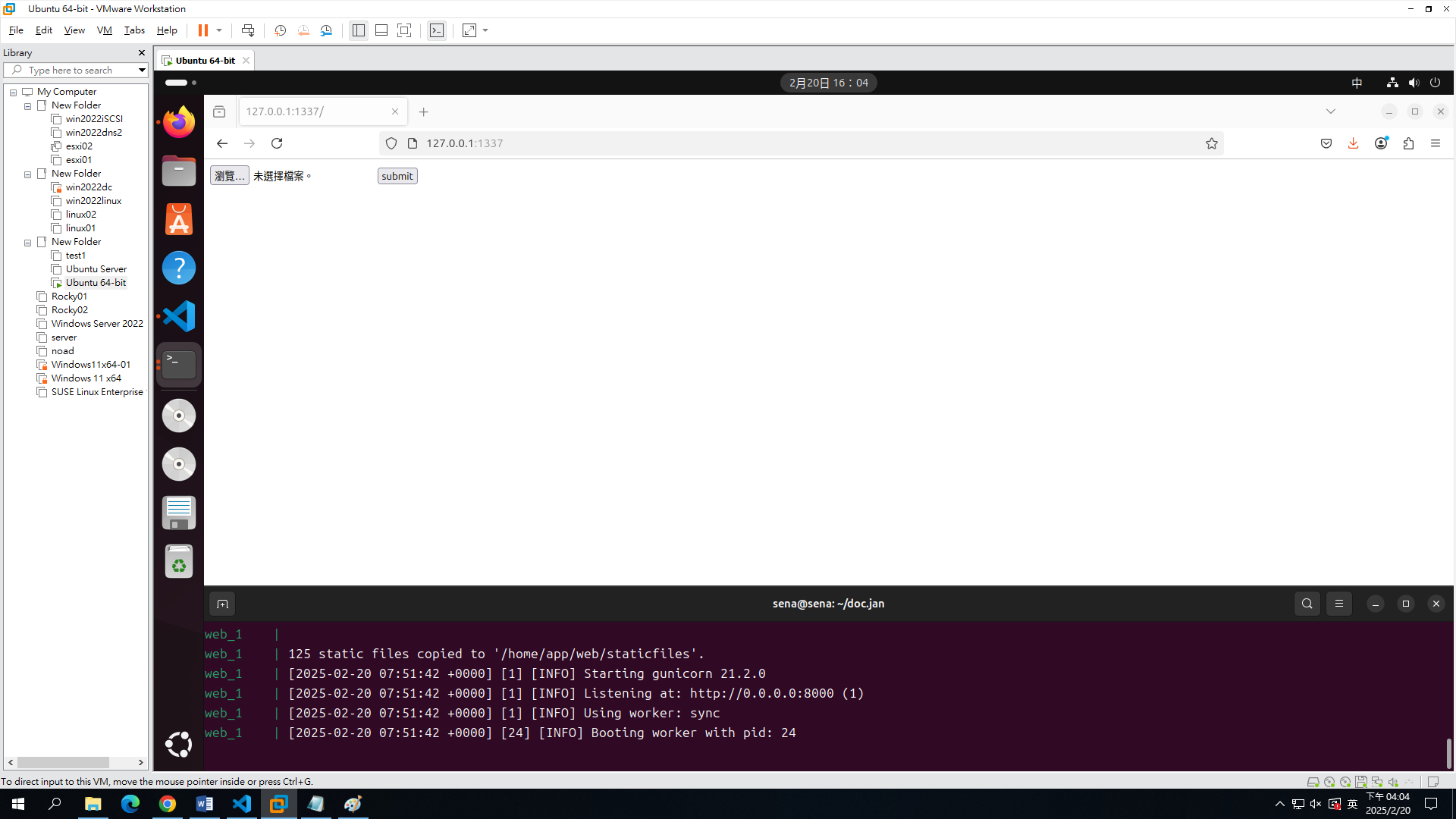


圖4-14 檔案上傳頁面

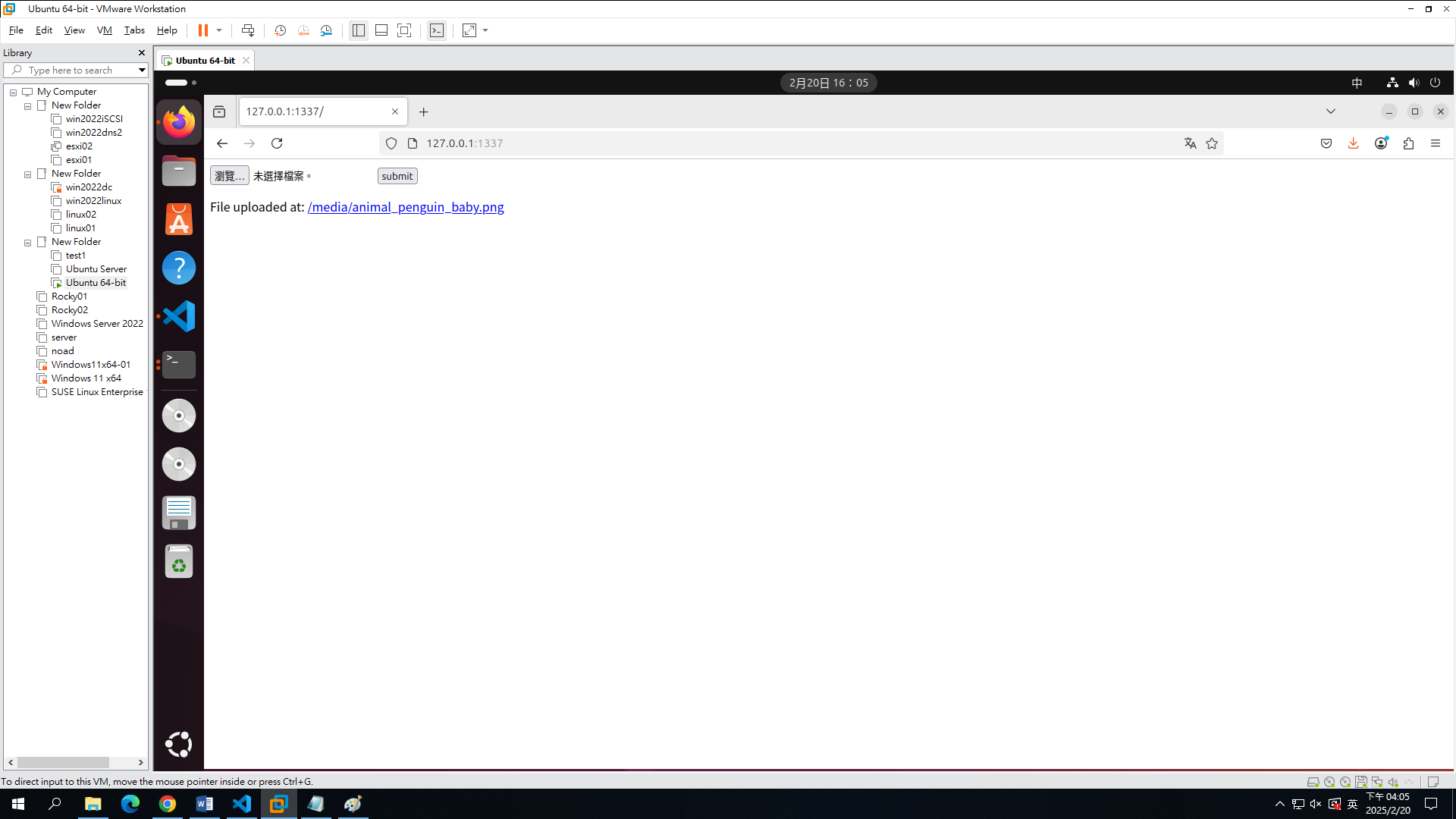


圖4-15 檔案上傳成功



圖4-16檢視上傳檔案成功

最後是不同環境移動測試，這邊原先的開發環境為ubuntu(圖4-17)，我將本資料傳送到rocky linux上，可看到在不同系統裡，一樣能夠使用本服務，不會有不相容情形。

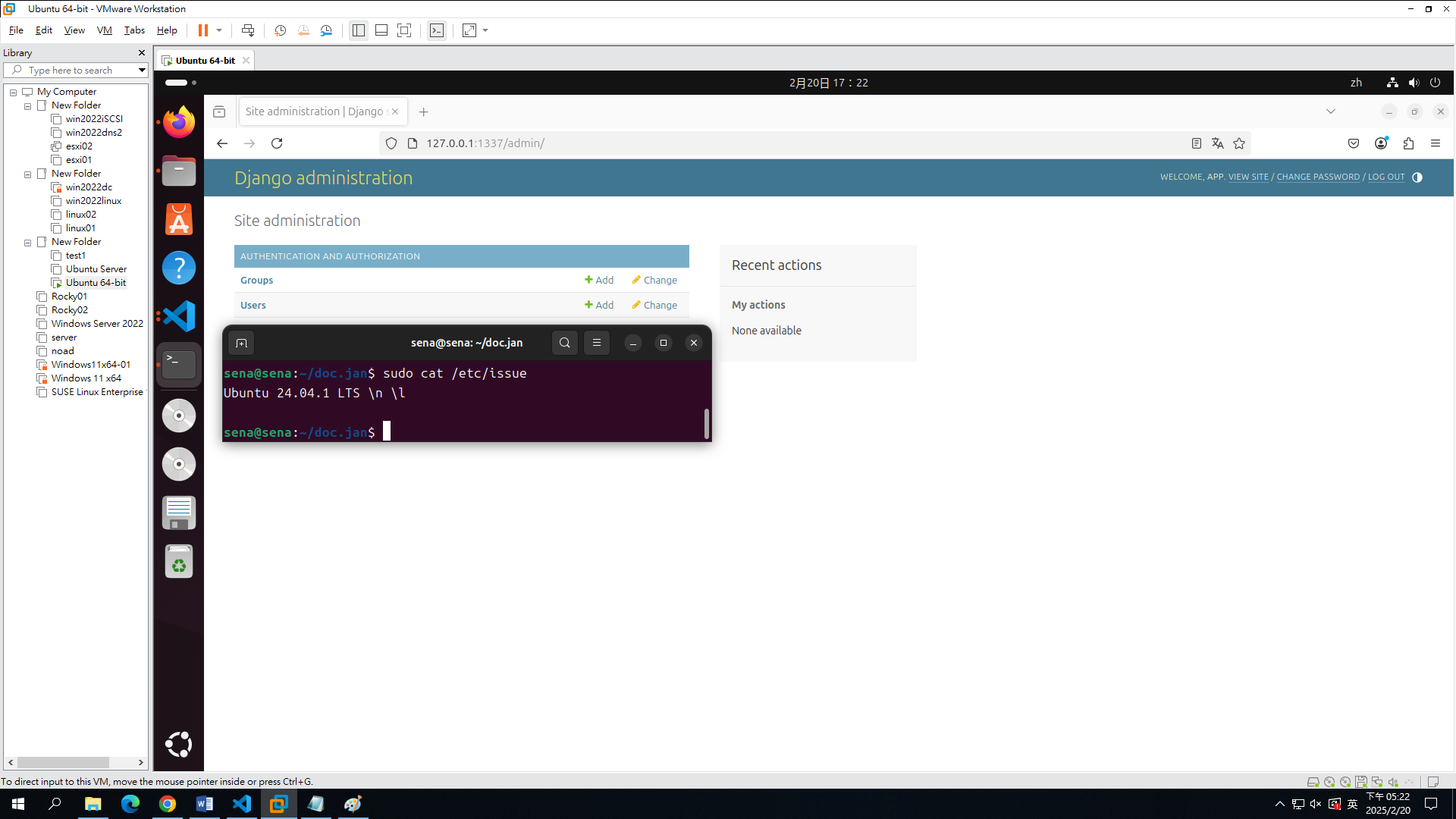
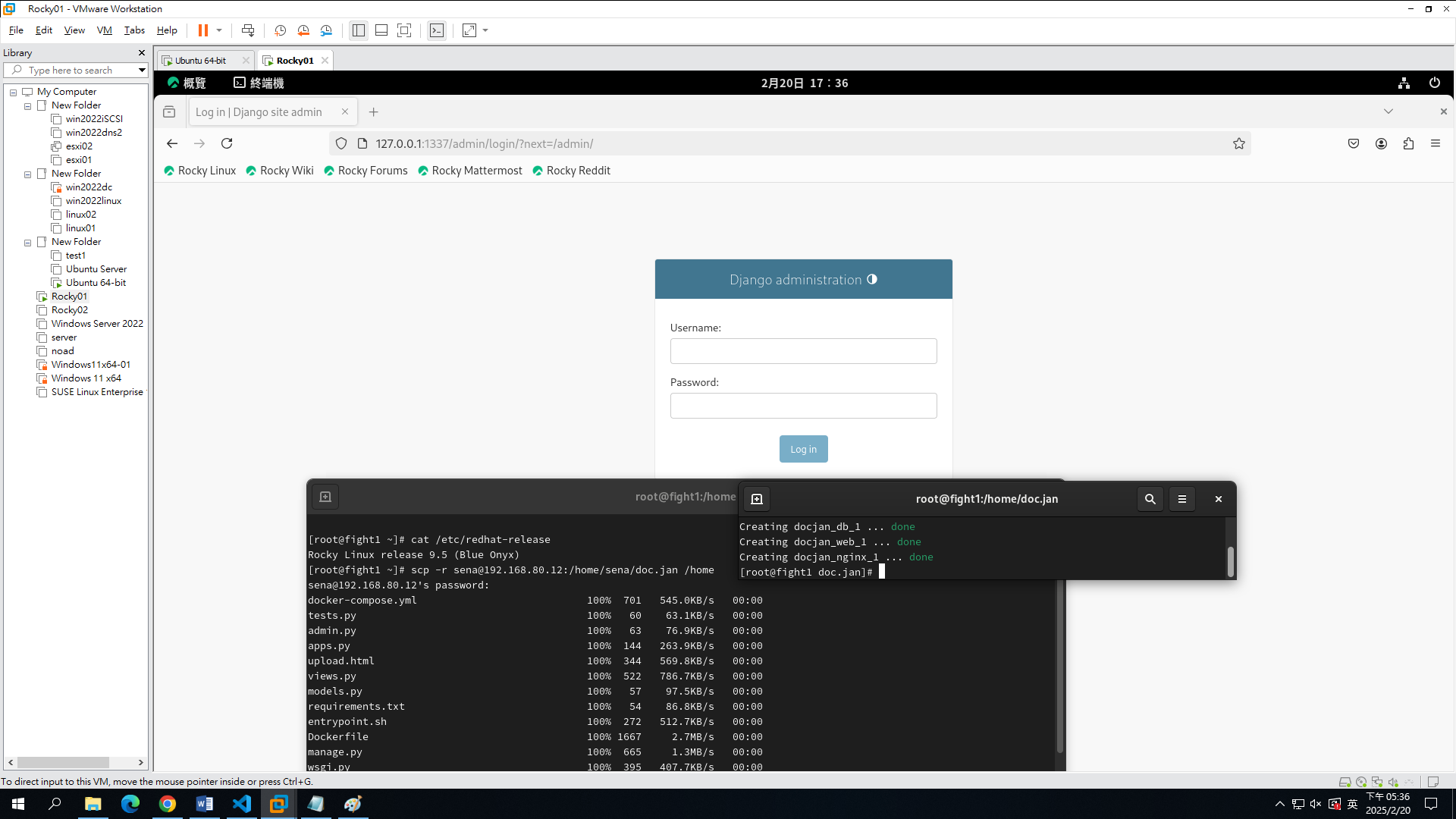


圖4-17開發環境為ubuntu



複製本次的專案

系統為rocky linux

三個容區皆啟動完成

圖4-18 rocky linux複製檔案並成功啟動登入驗證

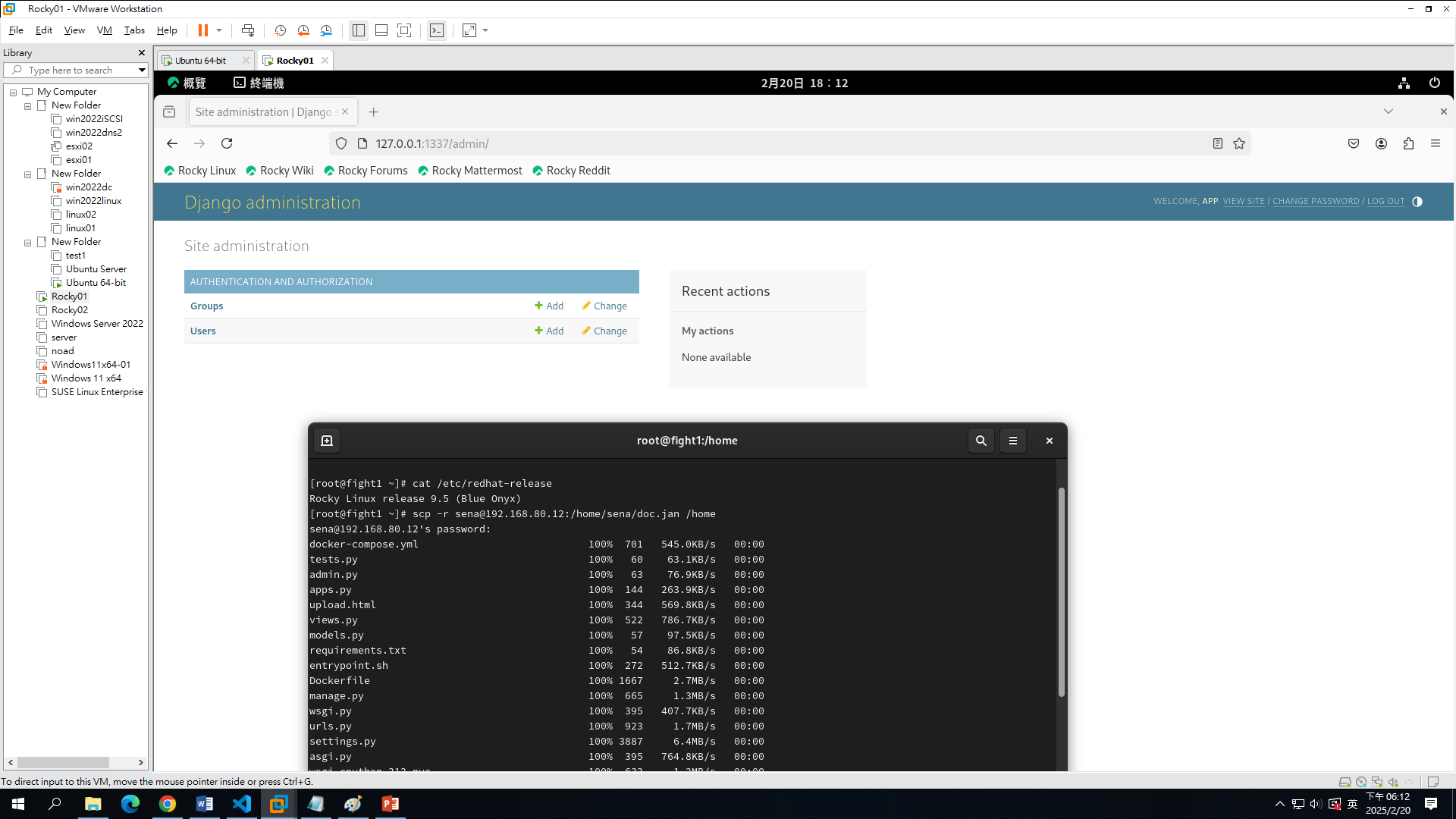


圖4-19 rocky linux登入成功

### 4.4.6結語

當我們需要架設多樣化的服務但不希望因為系統不同、版本不同或各個服務互相干擾造成整體服務無法運作，可以利用docker容器的技術將需要的服務打包在容器裡。如果官方的映像檔無法符合需求時可撰寫Dockerfile設定容器的環境以利服務運行。需要將容器連接可透過docker compose進行設定並在設定完成後一次啟動所有容器。

### 4.4.7 參考資料

Dockerizing Django with Postgres, Gunicorn, and Nginx

<https://testdriven.io/blog/dockerizing-django-with-postgres-gunicorn-and-nginx/>

docker-composeでDjango開発環境を構築する

<https://zenn.dev/dsonoda/articles/dbe14ca8af617ed85b1f>

docker-django-nginx-uwsgi-postgres-tutorial

<https://github.com/twtrubiks/docker-django-nginx-uwsgi-postgres-tutorial>