## **Create instance and security groups**

Refer these steps for making instances on AWS console.

[EC2-Jenkins\_server](https://scribehow.com/shared/EC2__Jenkins_Server_on_AWS__yWzjq6RQRJ6-jf1RnHlftw?referrer=workspace)

[EC2-Database\_server](https://scribehow.com/shared/EC2_Instance_for_database_On_AWS__TStuF_-2TpKp6Uozsqasiw?referrer=workspace)

[EC2-Backend\_server](https://scribehow.com/shared/EC2_Instance_for_backend_server_on_AWS__vrzyys_HSQyQmOIXwDeFBA?referrer=workspace)

Refer these steps for making security groups on AWS console.

[SG-Jenkins\_server](https://scribehow.com/shared/Security_Group_for_Jenkins_Server__Vd4tY8QdRW-nxazso-AK1w?referrer=workspace)

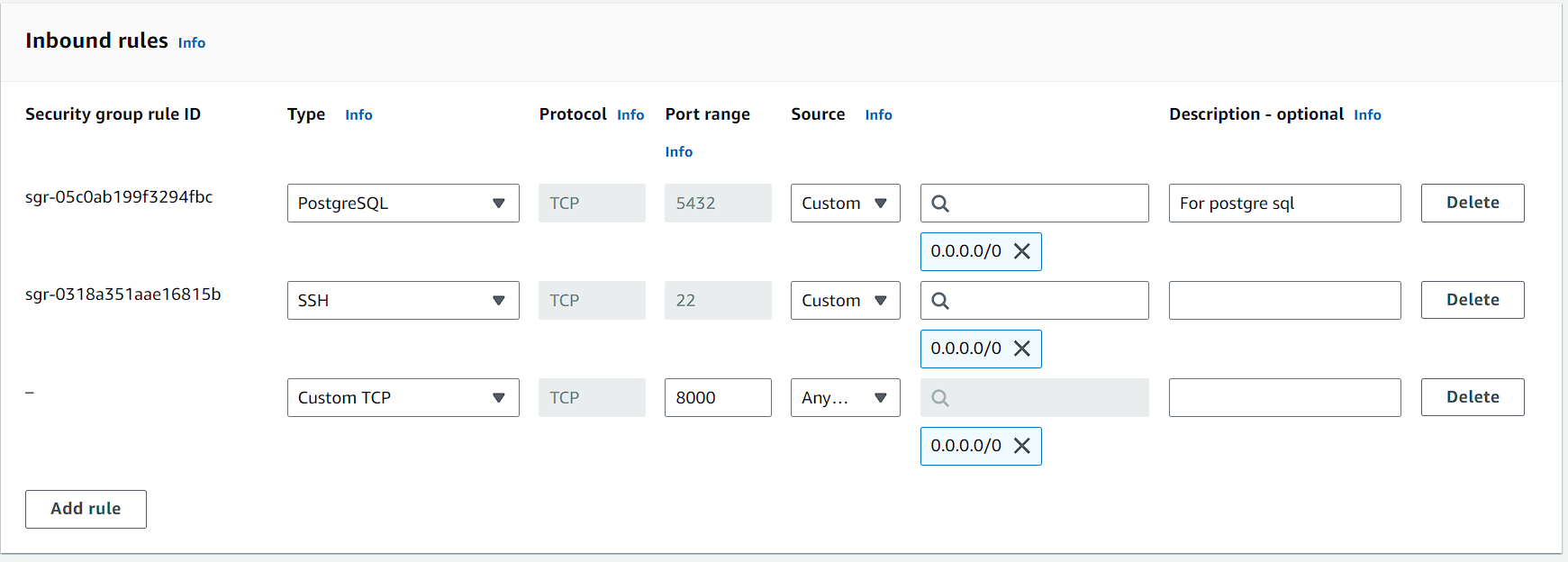
[SG-Backend\_server](https://scribehow.com/shared/Create_AWS_Security_Group_for_backend_Server__iZt8JPSjSSuqemvQIz_3SA?referrer=workspace)

## **PostgreSQL on Database Instance**

### **1. Set Up PostgreSQL on an Ubuntu 22.04 EC2 Instance**

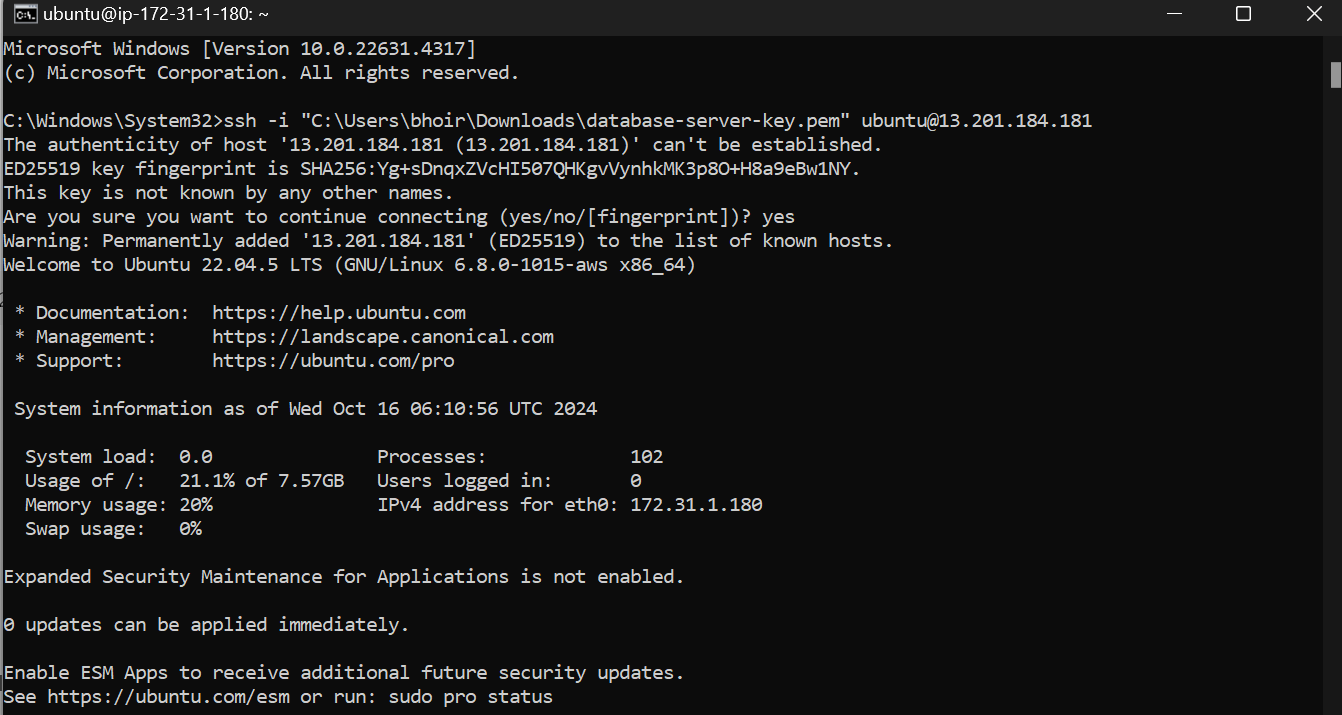
**Step 1: Launch EC2 Instance for PostgreSQL (Can refer above links)**

* Launch a new EC2 instance using Ubuntu 22.04 as the OS.
* Configure security group to allow connections on port 5432 (PostgreSQL default port).



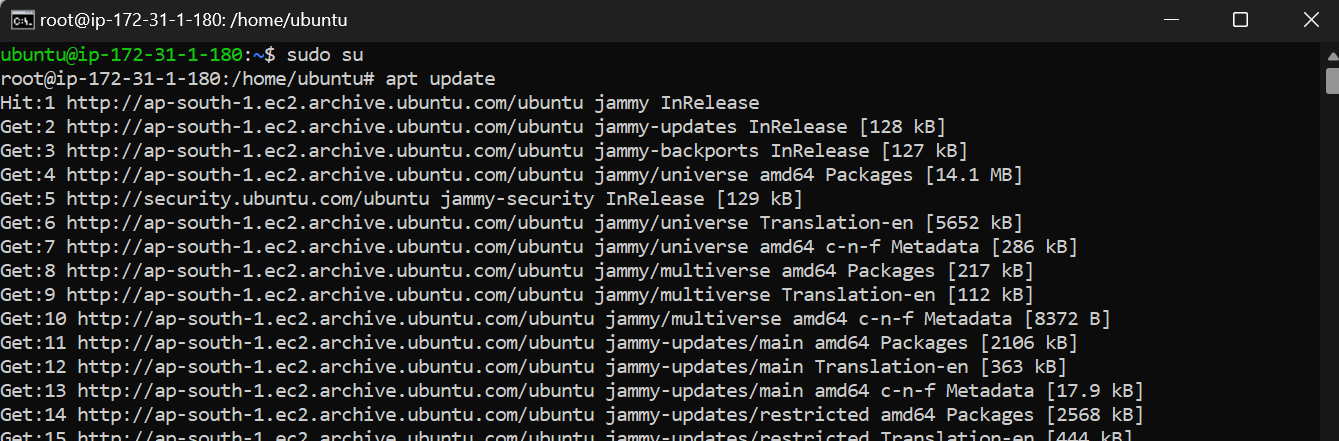
**Step 2: Connect to the EC2 Instance**

Connect to your instance using SSH.  
  
ssh -i your-key.pem ubuntu@your-ec2-public-ip



**Step 3: Switch to Root User**

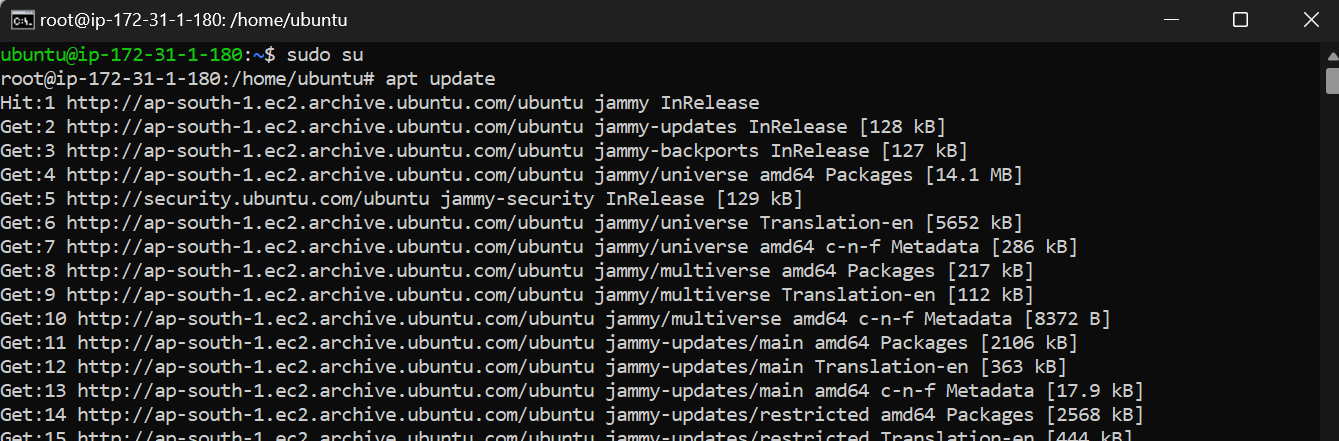
Once connected, switch to the root user.  
  
sudo su



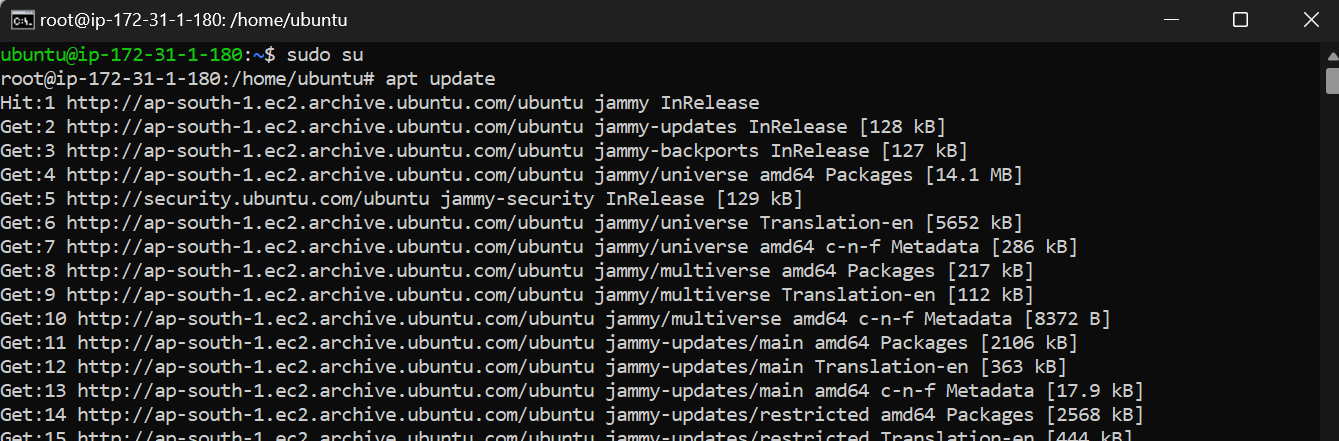
**Step 4: Install PostgreSQL**

Update your package list and install PostgreSQL.

apt update

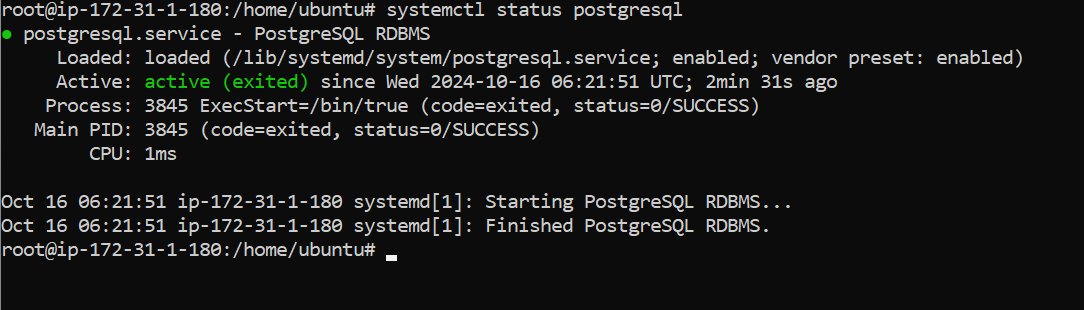


apt install postgresql postgresql-contrib



**Step 5: Verify PostgreSQL Installation**

After installation, check if PostgreSQL is running.  
  
systemctl status postgresql



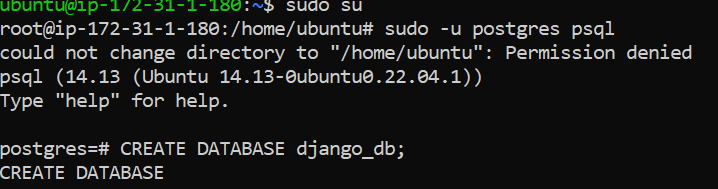
### **2. Configure PostgreSQL**

**Step 1: Switch to PostgreSQL User**

Switch to the postgres user, which is the default PostgreSQL user.  
  
su - postgres

**Step 2: Access PostgreSQL Shell (psql)**

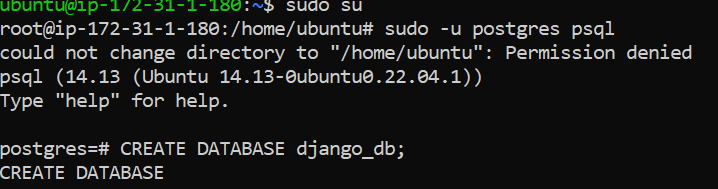
Open the PostgreSQL shell.  
  
psql



### **3. Create a Database and a User**

**Step 1: Create a New Database**

In the PostgreSQL shell, create a database for your Django application.  
  
CREATE DATABASE db\_name;



**Step 2: Create a New User**

Create a new user with a password.  
  
CREATE USER myappuser WITH PASSWORD 'securepassword';



**Step 3: Grant Privileges to the User**

Grant the necessary privileges on the database to the user.  
  
GRANT ALL PRIVILEGES ON DATABASE db\_name TO myappuser;



**Step 4: Exit the PostgreSQL Shell**

After setting up the user and database, exit the psql shell.  
  
\q

**Step 5: Allow External Connections (Optional)**

* By default, PostgreSQL only accepts connections from localhost. To allow external connections (from your Django EC2 instance):

1. Edit the PostgreSQL configuration file to allow connections:

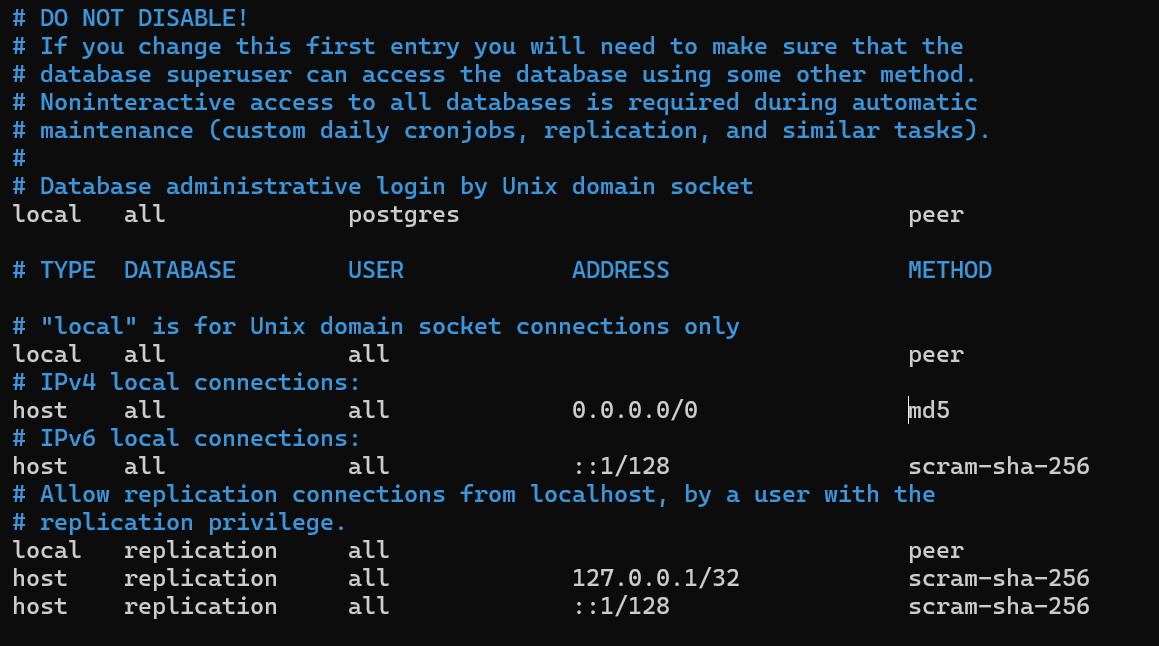
nano /etc/postgresql/14/main/postgresql.conf

Uncomment and change the listen\_addresses to accept connections from any IP:  
  
listen\_addresses = '\*'

1. Edit the pg\_hba.conf file to add your Django EC2 instance's IP address:

nano /etc/postgresql/14/main/pg\_hba.conf

Add the following line to allow connections from your Django EC2 instance:  
  
host all all your-django-ec2-ip/32 md5



1. Restart PostgreSQL for the changes to take effect:  
     
   systemctl restart postgresql

## **Django Application on EC2 Instance**

### **1. Launch EC2 Instance for Django**

**Step 1: Launch an EC2 Instance**

* Go to the AWS Console and launch an EC2 instance (Ubuntu 22.04 or similar).
* Configure the security group to allow traffic on:
  + **Port 22 (SSH)**: For remote access to your instance.
  + **Port 8000 (Django Default Port)**: For Django development server traffic.

**Step 2: Connect to the EC2 Instance**

After launching the instance, connect to it using SSH:

ssh -i your-key.pem ubuntu@your-ec2-public-ip

**Step 3: Switch to the Root User**

Once connected, switch to the root user.  
  
sudo su

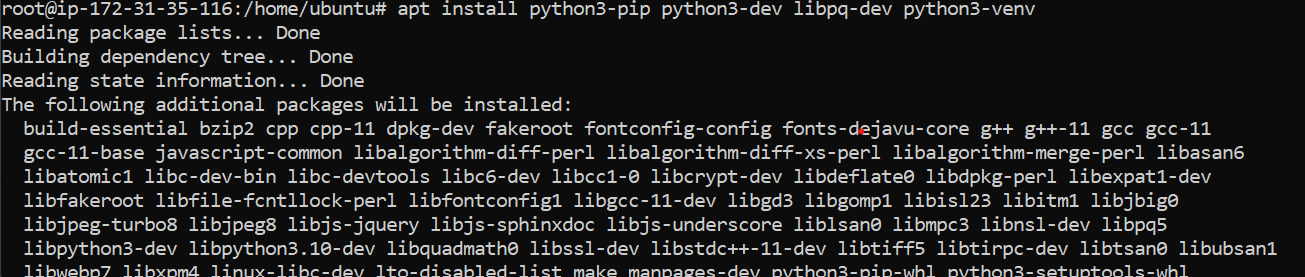
### **2. Install Dependencies**

**Step 1: Update Packages**

Update the package list and upgrade any existing packages.  
  
apt update && apt upgrade

**Step 2: Install Python, Pip, and Virtualenv**

Install Python, pip, and virtualenv to manage your Python environment.  
  
apt install python3-pip python3-venv

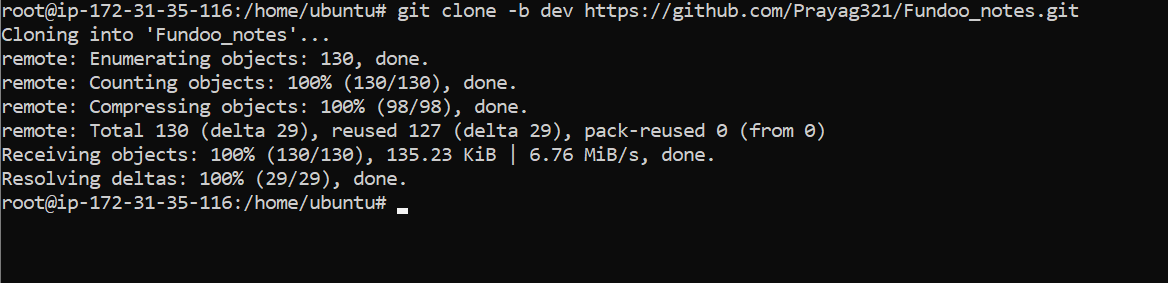


### **3. Set Up the Django Application**

**Step 1: Clone Your Django Project**

If your project is stored in a Git repository, clone it to your EC2 instance.  
  
git clone https://github.com/your-username/your-django-project.git

cd your-django-project



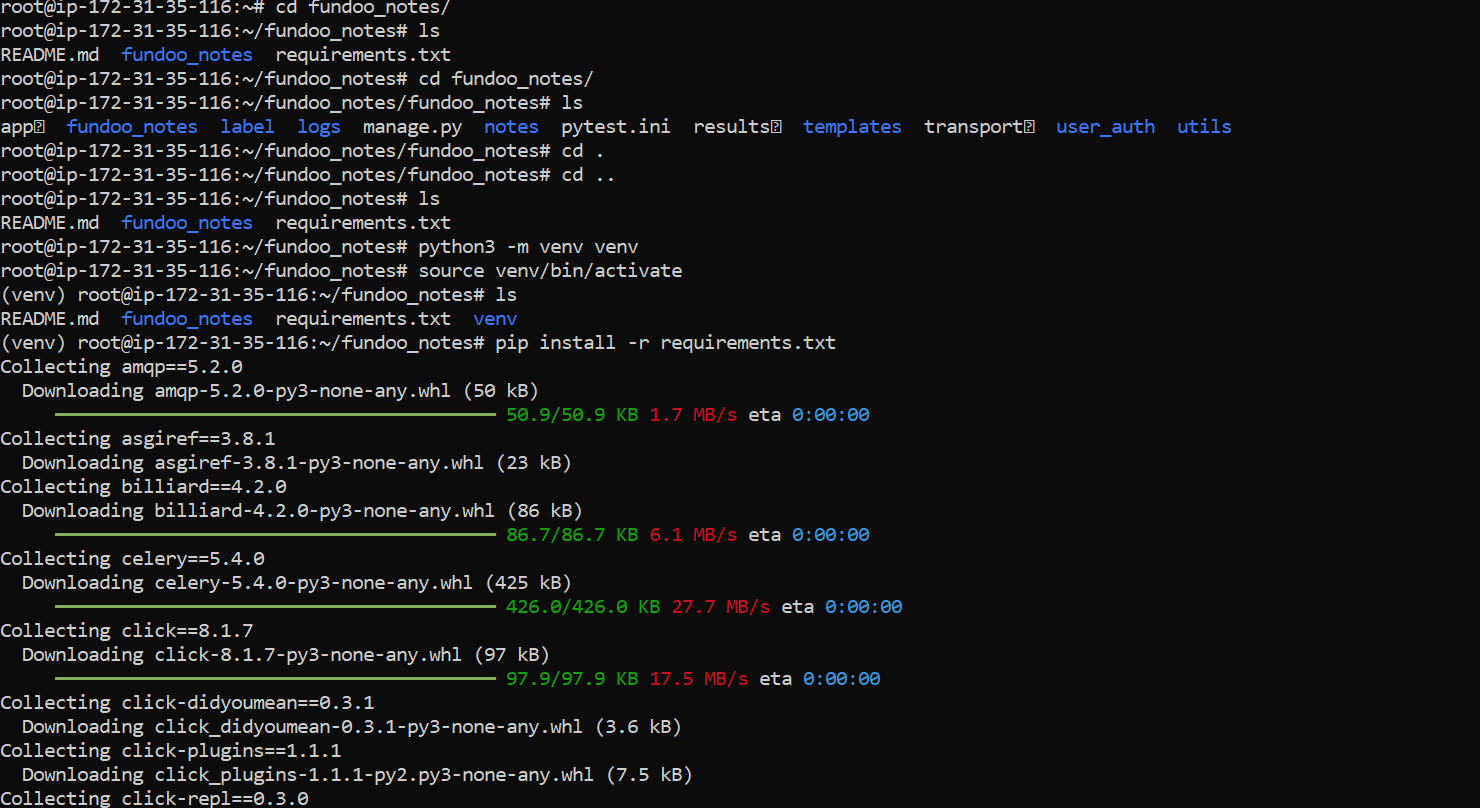
**Step 2: Create and Activate a Virtual Environment**

Create a Python virtual environment and activate it:  
  
python3 -m venv venv

source venv/bin/activate

**Step 3: Install Project Dependencies**

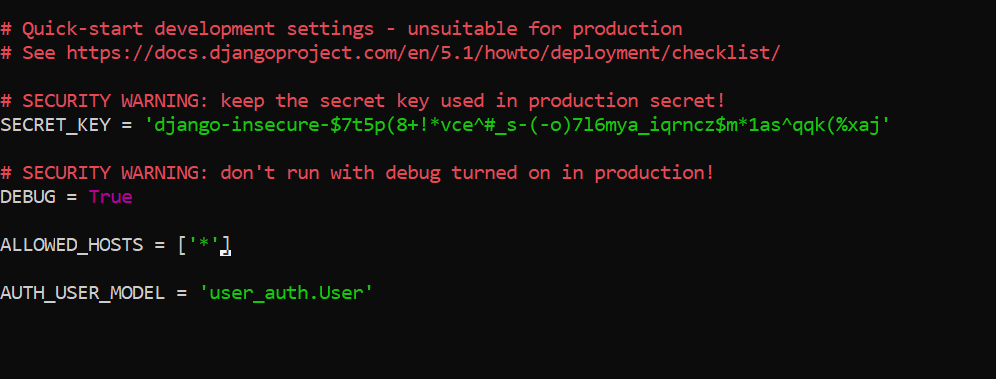
Install the required Python packages (Django and others) listed in your requirements.txt.  
  
pip install -r requirements.txt



### **4. Configure Django Settings**

**Step 1: Update Allowed Hosts**

In your Django settings.py file, update the ALLOWED\_HOSTS list to include your EC2 instance's public IP or domain:  
  
ALLOWED\_HOSTS = ['\*']



**Step 2: Configure the Database**

* Ensure that your DATABASES section is pointing to your PostgreSQL instance. If the database is running on another EC2 instance, add the hostname/IP and credentials.

Example DATABASES configuration in settings.py:

python

Copy code

DATABASES = {

'default': {

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

'NAME': 'your-db-name',

'USER': 'your-db-user',

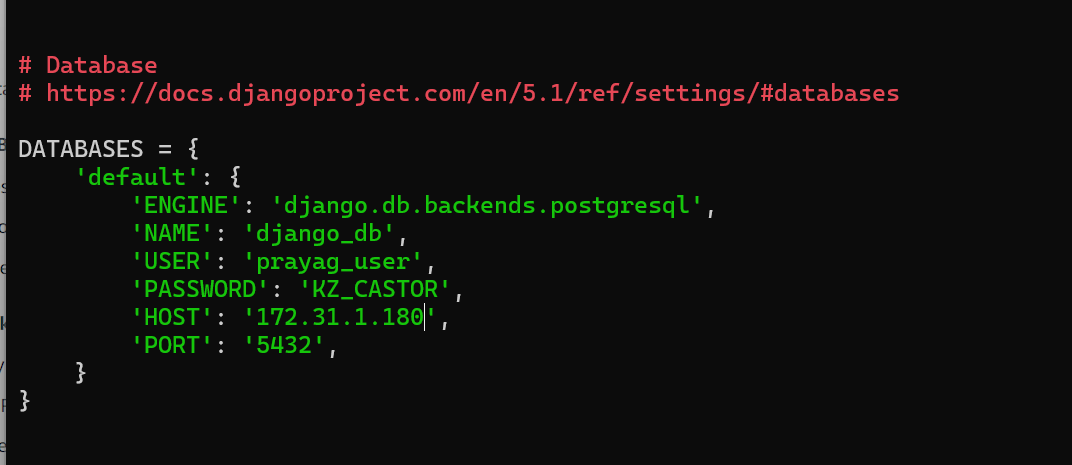
'PASSWORD': 'your-db-password',

'HOST': 'your-db-instance-ip', # IP address of PostgreSQL EC2 instance

'PORT': '5432',

}

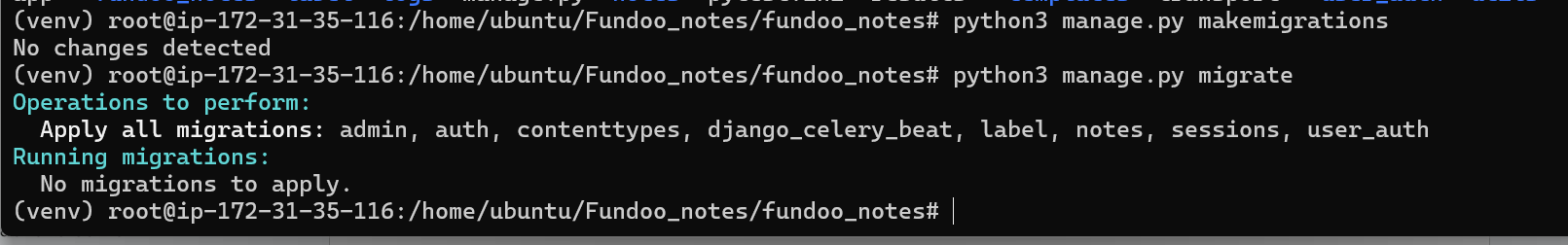
}



### **5. Set Up the Database**

**Step 1: Apply Migrations**

Run Django migrations to set up the database schema.  
  
python manage.py migrate



**Step 2: Create a Superuser (Optional)**

Create an admin user to access the Django admin panel.  
  
python manage.py createsuperuser

**Step 3: Collect Static Files**

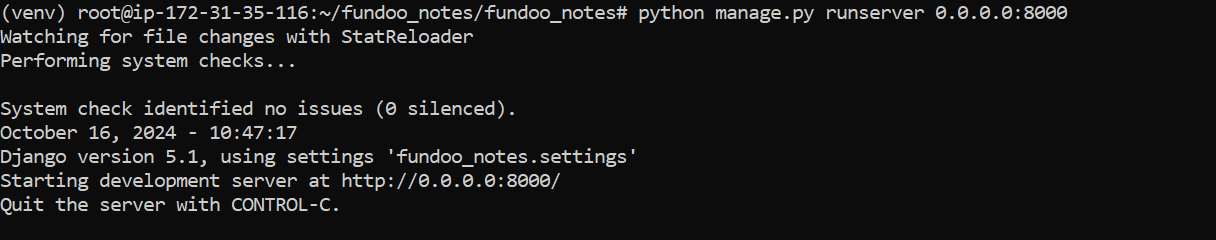
Collect static files for the project (if your project has any).  
  
python manage.py collectstatic

### **6. Run the Django Development Server**

**Step 1: Run Django Server**

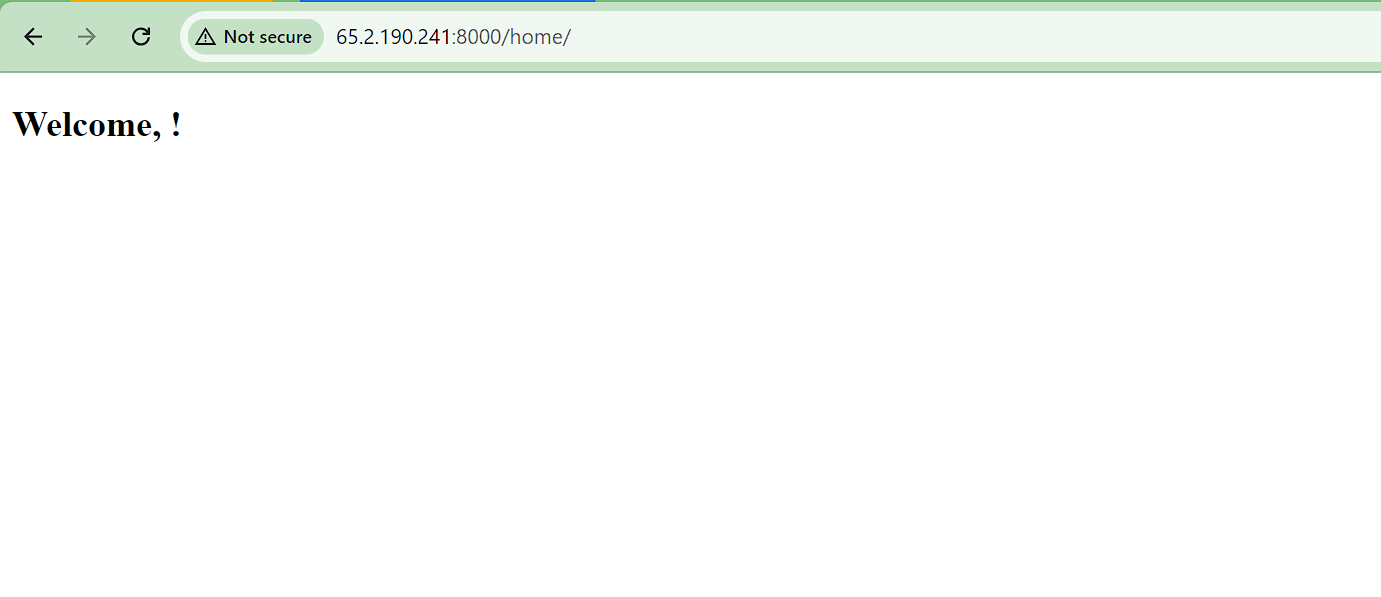
Run the Django development server on all available IPs (to make it accessible from the EC2 instance public IP). You need to use 0.0.0.0 as the host to allow external access:  
  
python manage.py runserver 0.0.0.0:8000

Now, your application should be accessible at:  
vbnet  
  
http://your-ec2-public-ip:8000

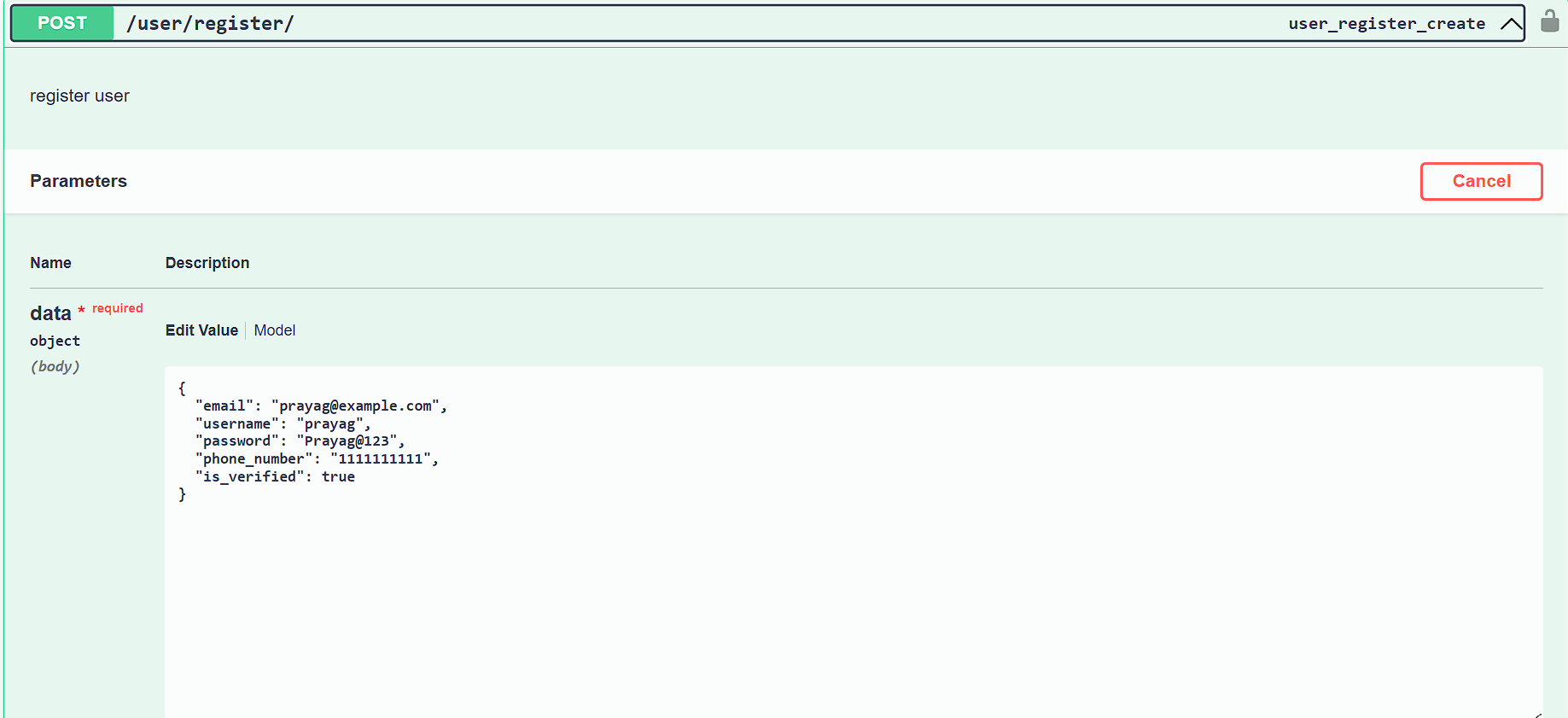


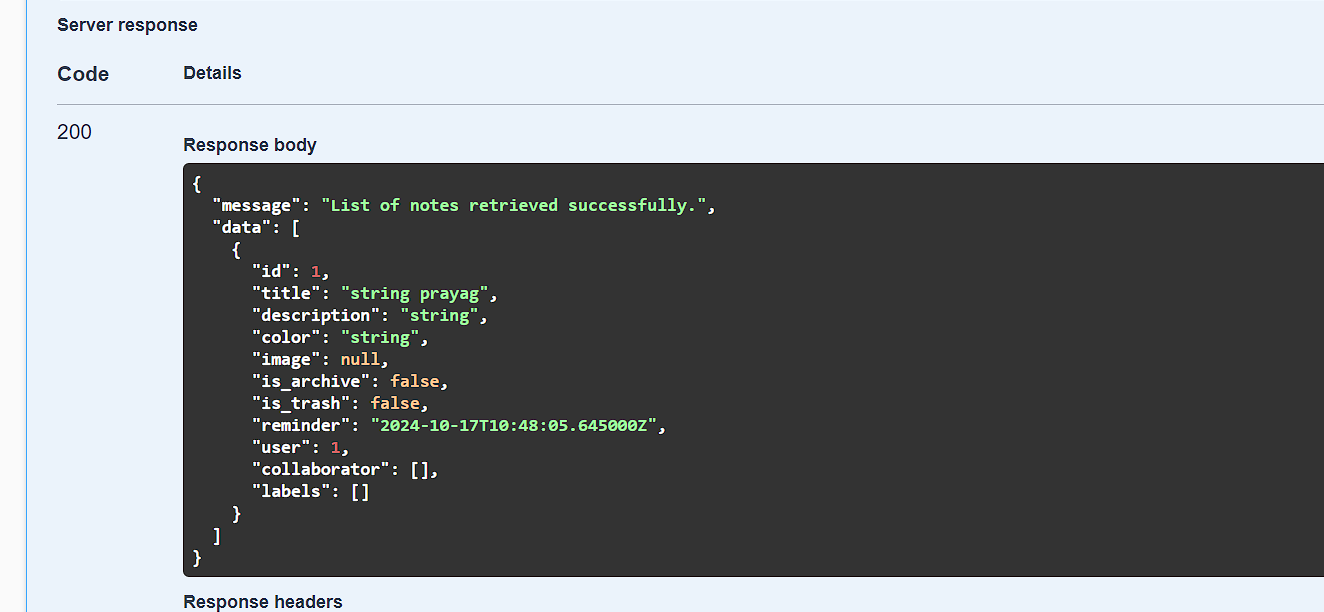
**Step 2: Access the Application**

* Open your browser and go to http://your-ec2-public-ip:8000. Your Django application should now be running on the EC2 instance.



* Test the APIs





### **7. Create a Systemd Service to Run Django as a Daemon**

To keep your Django application running even after closing the terminal or rebooting the EC2 instance, you can create a systemd service that will manage the application as a background daemon. Here’s how you can do that:

**1. Create a Service File**

We will create a custom service file for the Django project in the systemd directory.

1. Open the terminal on your EC2 instance.

Create a new service file with sudo in the /etc/systemd/system/ directory:  
  
sudo nano /etc/systemd/system/fundoo.service

**2. Define the Service**

In the service file, add the following configuration. Update the values like paths to your Python environment and Django project accordingly:

[Unit]

Description=Django Application

After=network.target

[Service]

User=your-username

Group=www-data

WorkingDirectory=/path/to/your/django/project

ExecStart=/path/to/your/python/env/bin/python manage.py runserver 0.0.0.0:8000

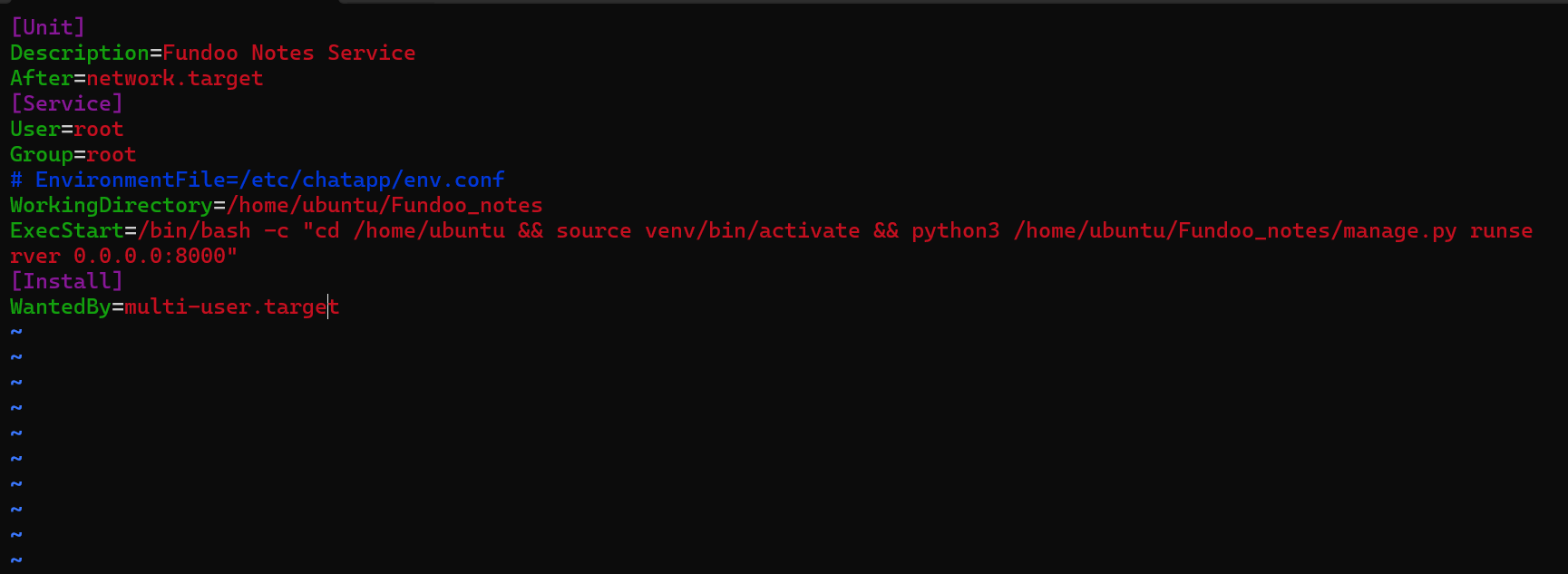
# Ensure that the application restarts if it crashes

Restart=always

RestartSec=5

[Install]

WantedBy=multi-user.target



* **User**: Replace your-username with the name of the user running the Django application (you can use whoami to find your username).
* **WorkingDirectory**: Path to your Django project folder where manage.py is located.
* **ExecStart**: Path to your Python environment's python binary (virtual environment, for example) and the manage.py runserver command to run Django.

**3. Reload systemd Daemon**

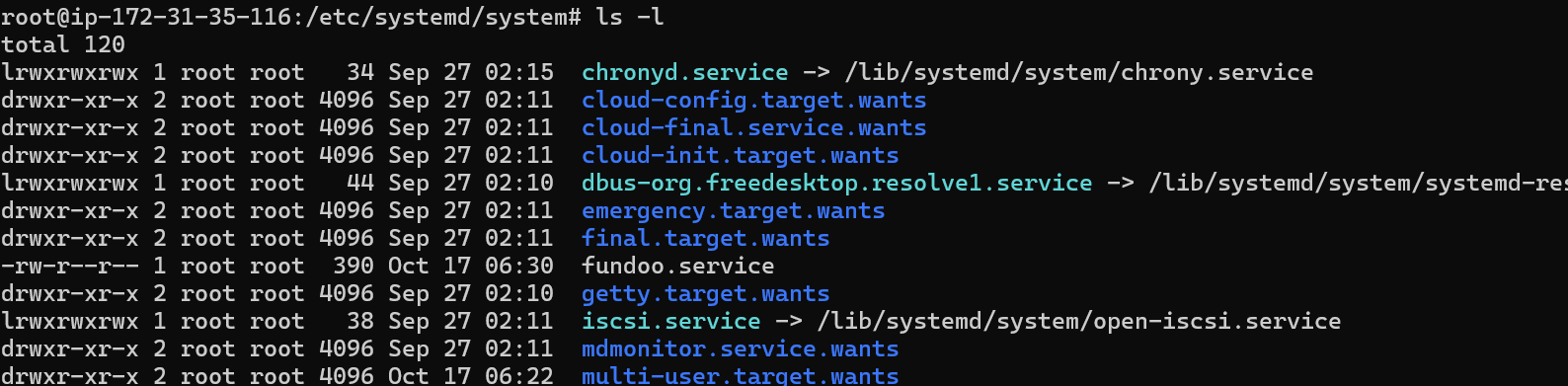
Once the service file is saved, reload the systemd daemon to recognize the new service:

sudo systemctl daemon-reload

**4. Start and Enable the Service**

To start the Django service and enable it to run automatically on boot, use the following commands:

* Start the service:  
    
  sudo systemctl start fundoo.service
* Enable the service to start on boot:  
    
  sudo systemctl enable fundoo.service



## **Jenkins on EC2 Instance**

### **Step 1: Launch an EC2 Ubuntu Instance (Can refer the above link)**

1. **Log in to the AWS Management Console.**
2. Navigate to **EC2 Dashboard** and click on **Launch Instance**.
3. Choose **Ubuntu Server 20.04 LTS (HVM), SSD Volume Type** from the list of Amazon Machine Images (AMI).
4. Select an instance type, like **t2.micro** (free tier eligible).
5. Configure the instance details, storage, and tags as needed. Leave the defaults for a basic setup.
6. **Configure the security group**:
   * Allow **port 22** for SSH.
   * Allow **port 8080** (Jenkins default port).
   * Allow **port 80** (if you want to set up a reverse proxy using Nginx or Apache for HTTP access).
7. **Launch the instance** and create a new key pair or use an existing one for SSH access.

### 

### 

### **Step 2: Connect to Your EC2 Instance**

After the EC2 instance is running, connect to it using SSH:  
  
ssh -i /path/to/your-key.pem ubuntu@your-instance-public-ip

### **Step 3: Update the Instance Packages**

Before installing Jenkins, update the package list and upgrade any outdated packages:

sudo apt update

sudo apt upgrade -y

### **Step 4: Install Java**

Jenkins requires Java to run. Install OpenJDK (Java Development Kit) on your Ubuntu instance:

sudo apt install openjdk-11-jdk -y

Check the installed Java version:

java -version

### **Step 5: Add the Jenkins Repository**

Refer this - [Debian Jenkins Packages](https://pkg.jenkins.io/debian-stable/)

Jenkins is not available in the default Ubuntu package repositories, so you need to add the Jenkins repository to your system:

**Add the Jenkins GPG key**:  
  
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

**Add the Jenkins repository**:  
  
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

### **Step 6: Install Jenkins**

Update the package list to include the Jenkins repository:  
  
sudo apt update

Install Jenkins:  
  
sudo apt install jenkins -y

### **Step 7: Start and Enable Jenkins**

Once Jenkins is installed, start the Jenkins service and enable it to run on system boot:

sudo systemctl start jenkins

sudo systemctl enable jenkins

### **Step 8: Adjust the Firewall (can be done in aws security group)**

Ensure that your instance's firewall allows traffic on **port 8080**:

Add a rule to allow Jenkins traffic on port 8080:  
  
  
sudo ufw allow 8080

Allow OpenSSH (port 22) and enable the firewall:  
  
sudo ufw allow OpenSSH

sudo ufw enable

sudo ufw status

### **Step 9: Access Jenkins**

1. Open your browser and navigate to:  
   vbnet  
     
   http://your-instance-public-ip:8080
2. You will see the Jenkins **Unlock Jenkins** screen. To unlock Jenkins, you need to retrieve the initial admin password:  
     
   sudo cat /var/lib/jenkins/secrets/initialAdminPassword
3. Copy the password and paste it into the Jenkins setup page to proceed with the installation.

**Connecting Jenkins with Backend instance and creating freestyle job and CI/CD Pipeline**

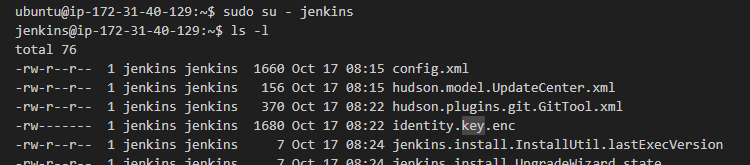
### **Step 1: Generate RSA Key Pair on Jenkins Server**

**Connect to your Jenkins EC2 instance** via SSH.

ssh -i path\_to\_your\_key.pem ec2-user@your\_ec2\_public\_dns

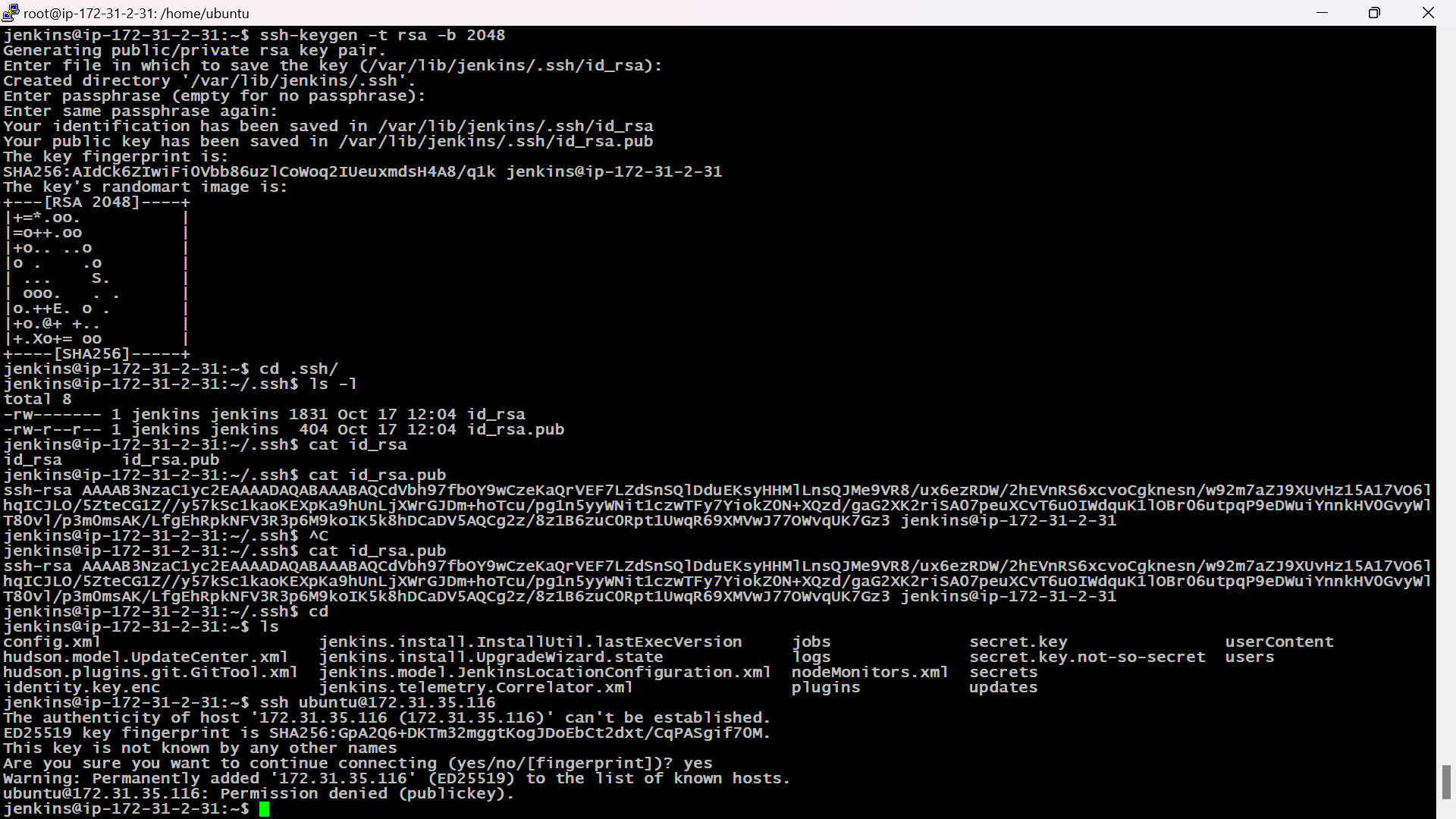
**Switch user to jenkins**

sudo su - jenkins



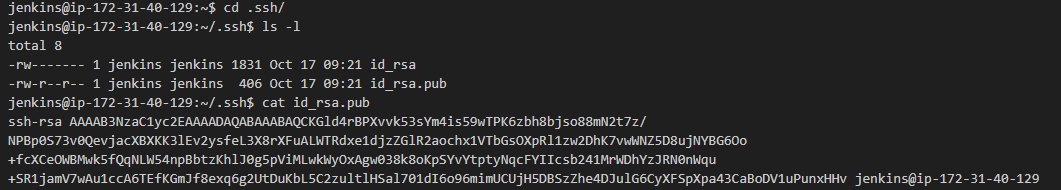
**Generate the RSA key pair:**

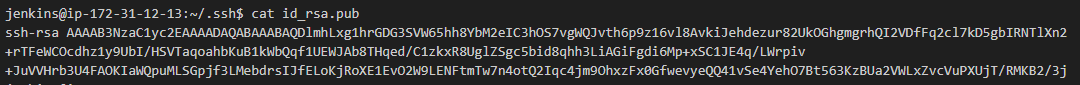
ssh-keygen -t rsa -b 2048



### **Step 2: Copy the Public Key to the Backend Server**

**Get the public key**:  
  
cat ~/.ssh/id\_rsa.pub



****

**Connect to your backend server instance** via SSH and **Add the public key to the authorized\_keys**:

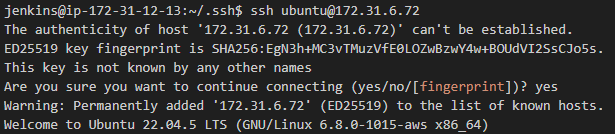
~/.ssh/authorized\_keys

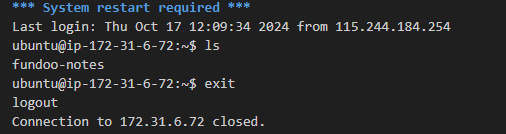
nano ~/.ssh/authorized\_keys

Paste the public key you copied from the Jenkins instance. and Save the file and exit.

### **Step 3: Test SSH Connection from Jenkins**

**Back on the Jenkins instance**, test the SSH connection to the backend server:  
  
ssh ubuntu@backend-server-private-ip



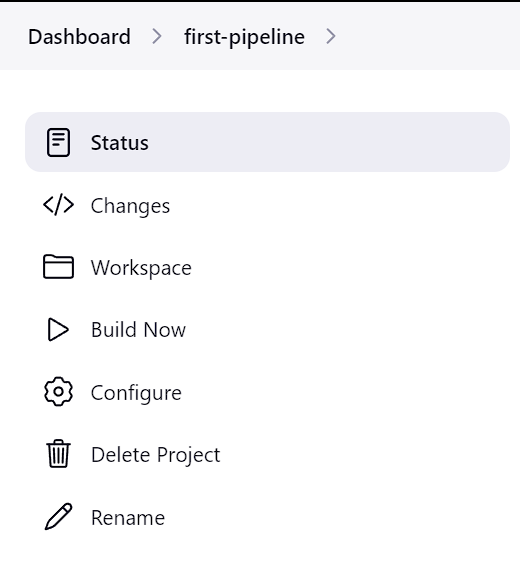


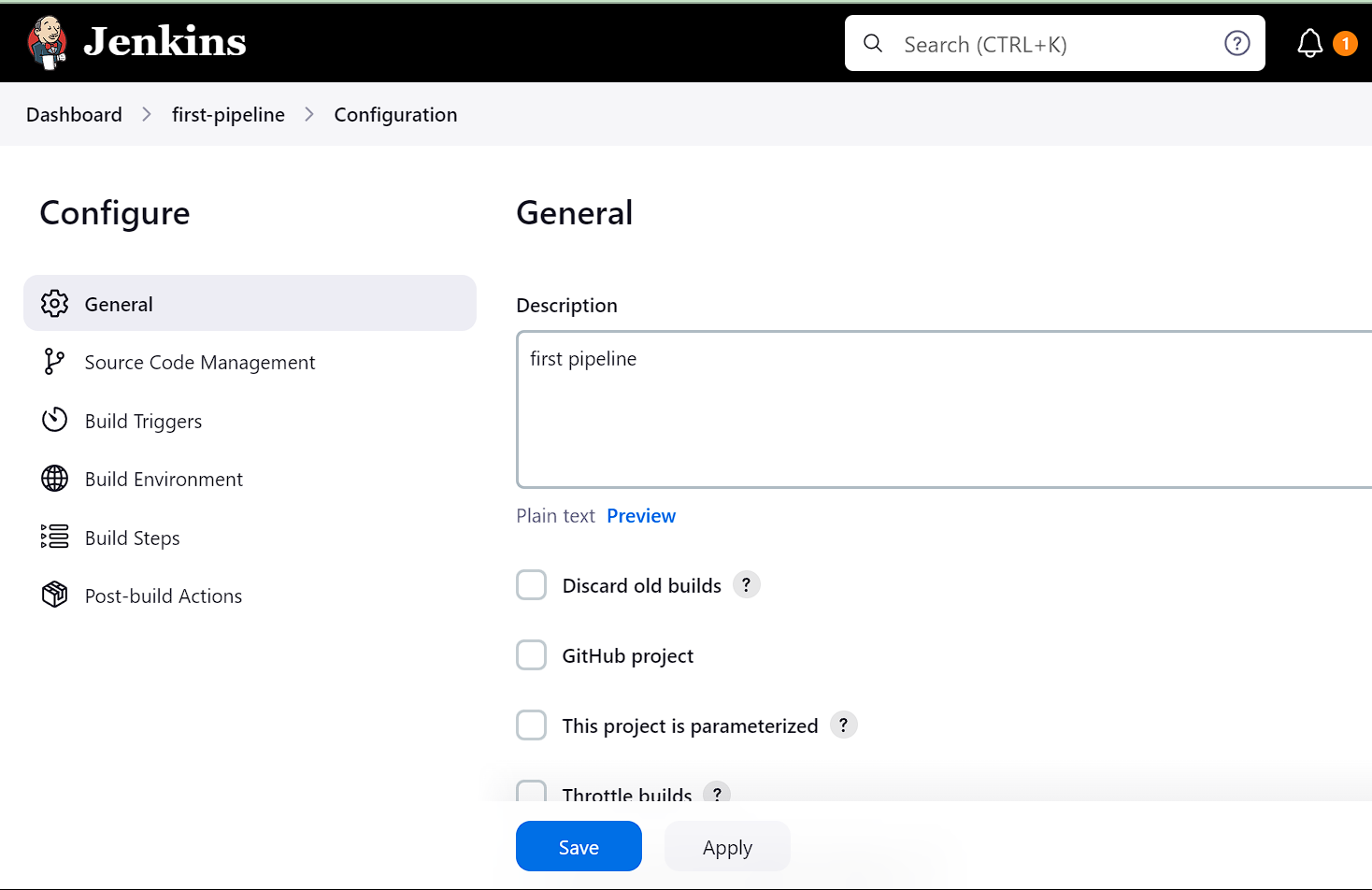
## Create a New Freestyle Project in Jenkins

* From the Jenkins dashboard, click on New Item.
* Enter a name for your job (e.g., DjangoAppBuild) and Select Freestyle project.

### **Configure Project**

* In the job configuration page, scroll to General





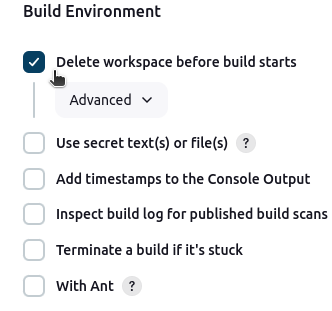
### **Configure Git repository**

* In the job configuration page, scroll down to Source Code Management.
* Select Git.
* Enter your repository URL (e.g., git@github.com:username/repo.git).
* If needed, configure credentials by clicking on Add next to Credentials.

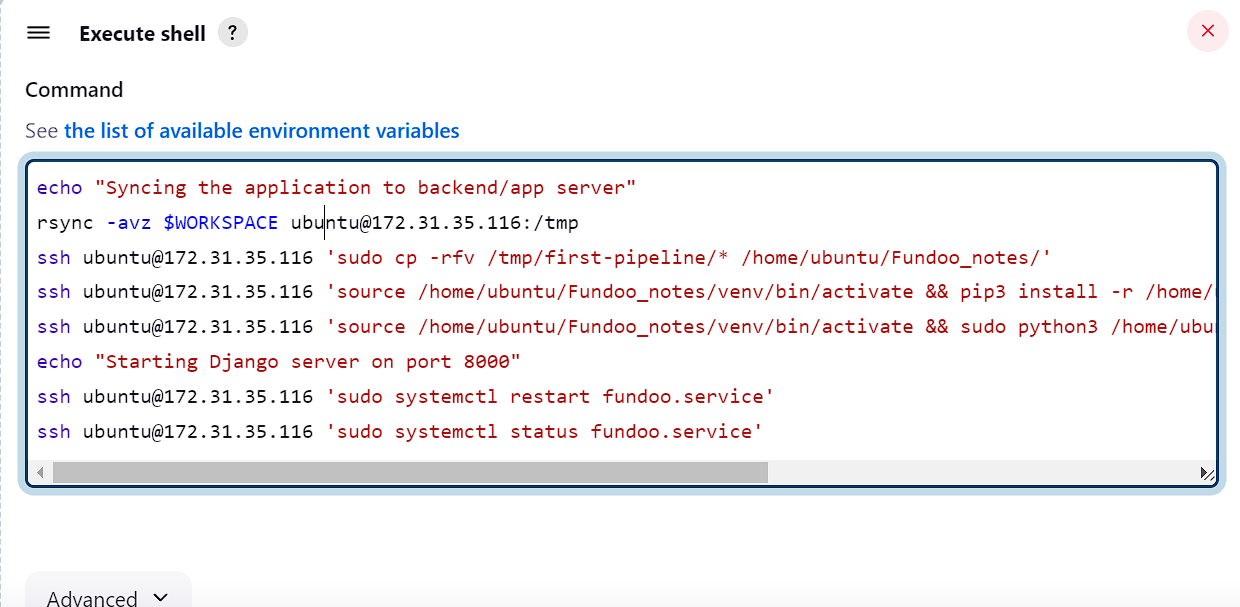
### **Configure Build Steps**

Here we configure our build

Select Discard Old builds

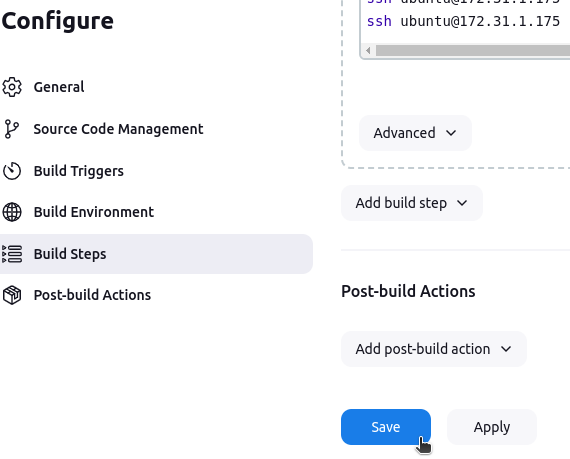


Build Steps

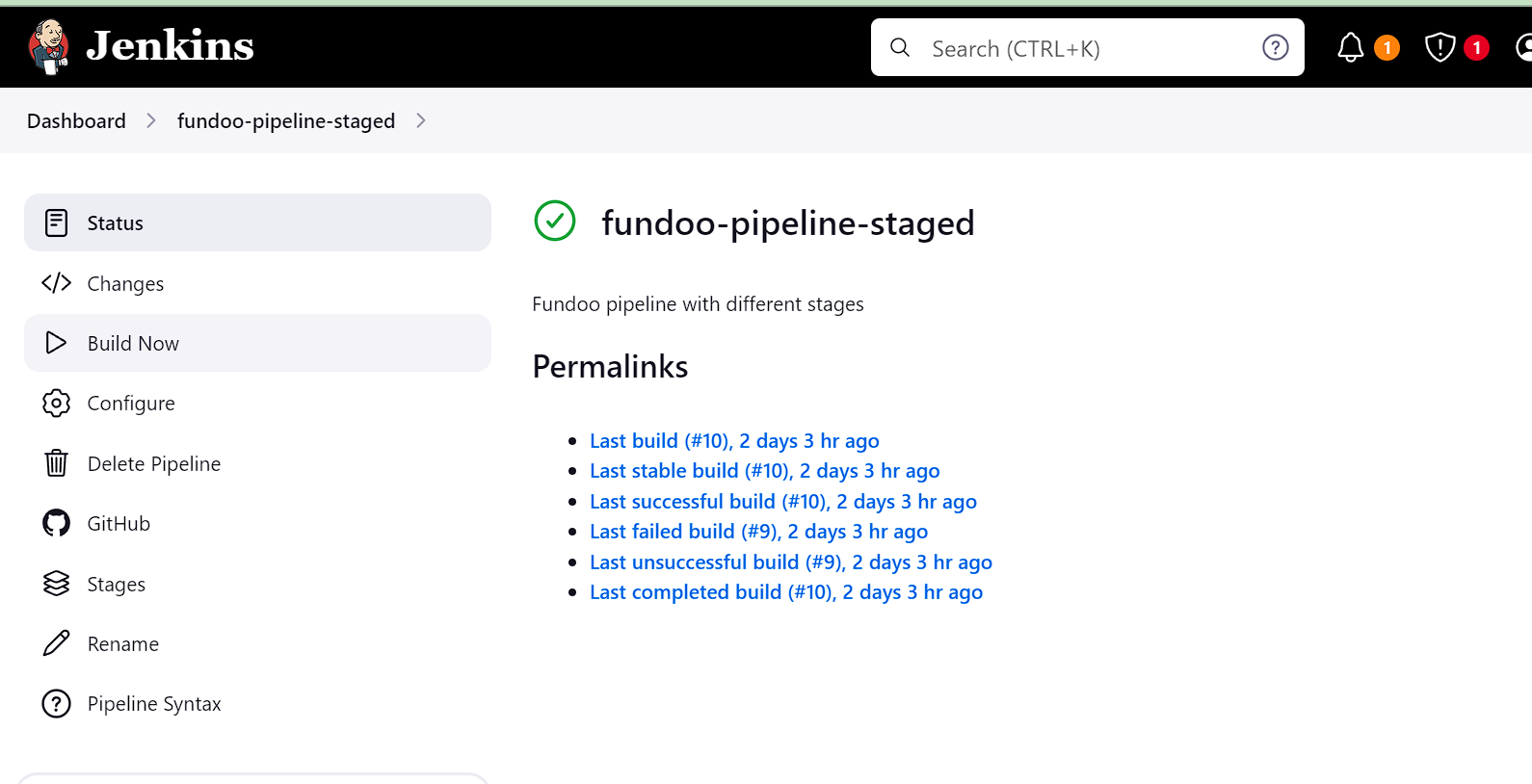


### **Save and Build**

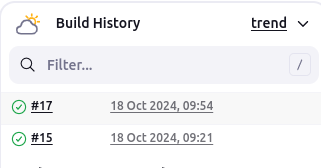
* Click on the Save button at the bottom of the configuration page.



* To run your job, go back to the job page and click on Build Now.



* Monitor the build process by clicking on the build number in the build history.



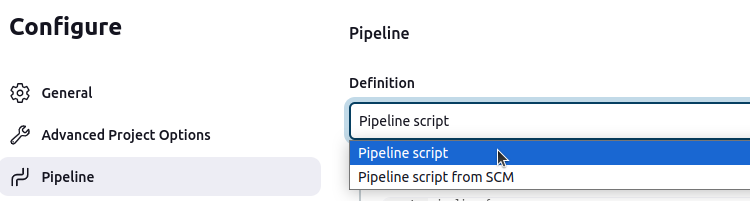
## **Create a New Pipeline Project in Jenkins**

### **Create New Job**

* From the Jenkins dashboard, click on New Item.
* Enter a name for your job (e.g., DjangoAppPipeline).
* Select Pipeline and click OK.

### **Configure Pipeline Script**

* In the job configuration page, scroll down to the Pipeline section.
* Choose Pipeline script from the Definition dropdown.



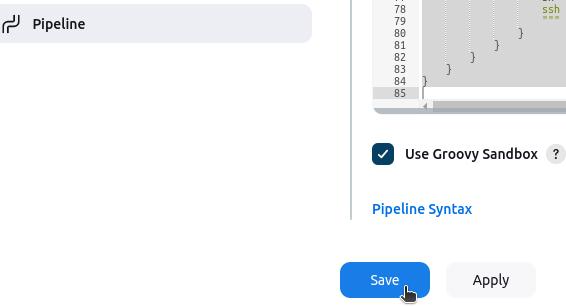
### **Enter Pipeline Script**

Here’s the basic pipeline script for our Django application

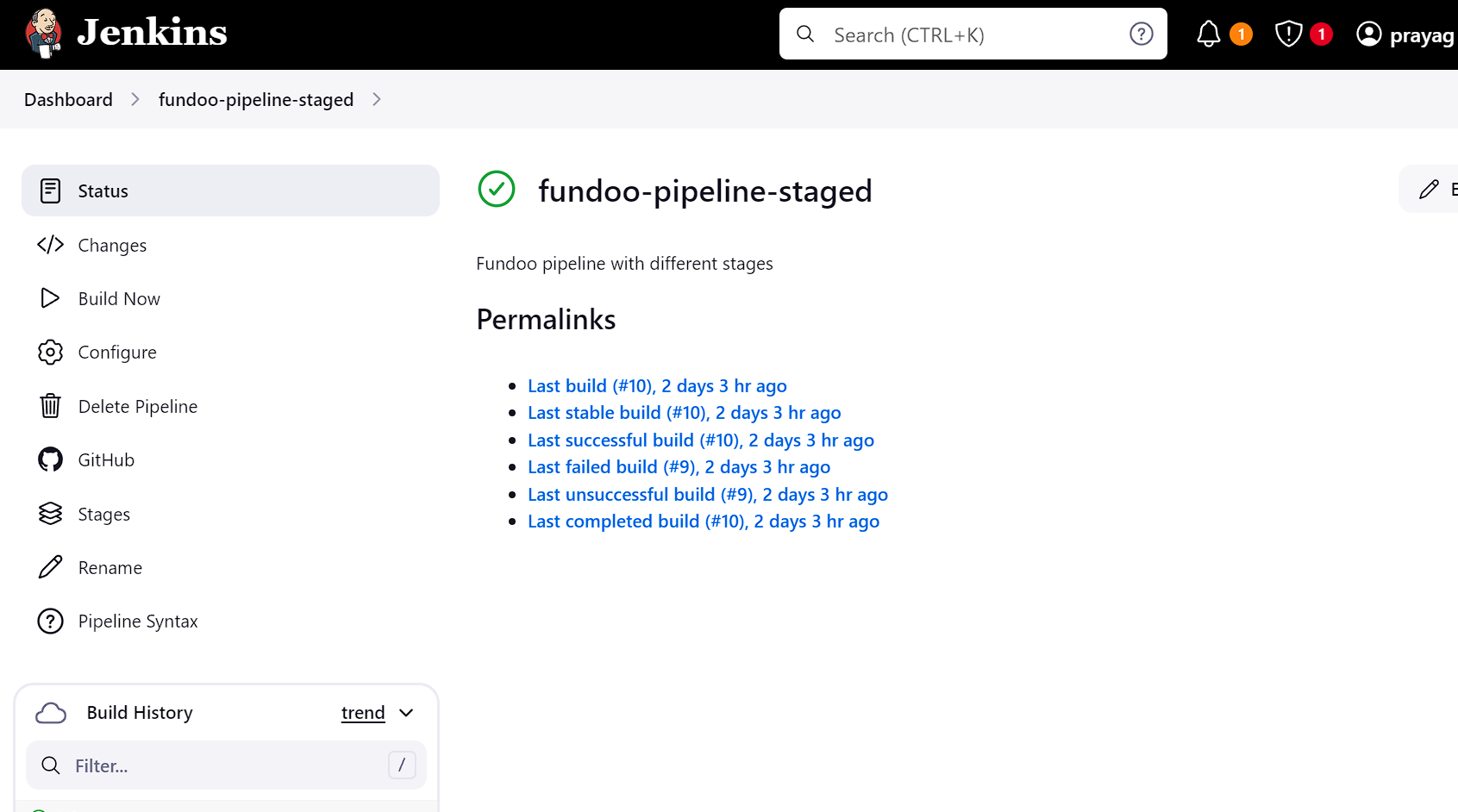
| **pipeline {**  **agent any**  **environment {**  **SERVER\_IP = '172.31.35.116'**  **APP\_DIR = '/home/ubuntu/Fundoo\_notes/'**  **TMP\_DIR = '/tmp/fundoo-pipeline-staged/'**  **REPO\_URL = 'https://github.com/Prayag321/Fundoo\_notes.git' // Replace with actual repository URL**  **BRANCH = 'pre-dev' // Replace with the desired branch name**  **}**  **stage('Preparation') {**  **steps {**  **echo 'Syncing the application to backend/app server'**  **sh """**  **rsync -avz $WORKSPACE root@${SERVER\_IP}:/tmp**  **"""**  **}**  **}**  **stage('Deploy Application') {**  **steps {**  **echo 'Copying files to the application directory'**  **sh """**  **ssh root@${SERVER\_IP} 'sudo cp -rfv ${TMP\_DIR}\* ${APP\_DIR}'**  **"""**  **}**  **}**  **stage('Install Dependencies') {**  **steps {**  **echo 'Installing dependencies in the virtual environment'**  **sh """**  **ssh root@${SERVER\_IP} '**  **source ${APP\_DIR}/venv/bin/activate &&**  **pip3 install -r ${APP\_DIR}/requirements.txt &&**  **pip3 install django celery &&**  **python3 ${APP\_DIR}/fundoo\_notes/manage.py migrate**  **'**  **"""**  **}**  **}**  **stage('Restart Application') {**  **steps {**  **echo 'Restarting Django application'**  **sh """**  **ssh root@${SERVER\_IP} 'sudo systemctl restart fundoo.service'**  **"""**  **}**  **}**  **stage('Check Application Status') {**  **steps {**  **echo 'Checking the status of the application service'**  **sh """**  **ssh root@${SERVER\_IP} 'sudo systemctl status fundoo.service'**  **"""**  **}**  **}**  **}**  **post {**  **always {**  **echo 'Pipeline completed'**  **}**  **success {**  **echo 'Pipeline completed successfully'**  **}**  **failure {**  **echo 'Pipeline failed'**  **}**  **}**  **}** |
| --- |

### **Save and Build**

1. Click on the Save button at the bottom of the configuration page.



1. To run your job, go back to the job page and click on Build Now.



1. Monitor the build process by clicking on the build number in the build history.

